A Branch-and-Price Algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows

G. Gutiérrez-Jarpa, G. Desaulniers, G. Laporte, V. Marianov

G–2009–19

March 2009
A Branch-and-Price Algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows

Gabriel Gutiérrez-Jarpa
Pontificia Universidad Católica de Chile, Santiago, Chile
and Department of Industrial Engineering
Universidad Católica de la Santísima Concepción, Concepción, Chile
ggutierrez@ucsc.cl

Guy Desaulniers
GERAD and Département de mathématiques et de génie industriel
École Polytechnique de Montréal, Montréal, Canada H3C 3A7
guy.desaulniers@gerad.ca

Gilbert Laporte
GERAD, CIRRELT and HEC Montréal
3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7
gilbert@crt.umontreal.ca

Vladimir Marianov
Department of Electrical Engineering
Pontificia Universidad Católica de Chile, Santiago, Chile
marianov@ing.puc.cl

March 2009

Les Cahiers du GERAD
G–2009–19

Copyright © 2009 GERAD
Abstract

In the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows, the set of customers is the union of delivery customers and pickup customers. A fleet of identical capacitated vehicles based at the depot must perform all deliveries and profitable pickups while respecting time windows. The objective is to minimize routing costs, minus the revenue associated with the pickups. Three versions of the problem are considered according to the order imposed on deliveries and pickups. An exact branch-and-price algorithm is developed for the problem. Computational results are reported for instances containing up to 100 customers.

Key Words: Vehicle routing problem, deliveries, selective pickups, time windows, backhauls, branch-and-price, column generation, shortest paths with resources.

Résumé

Dans le problème de tournées de véhicules avec livraisons, cueillettes sélectives et fenêtres de temps, l'ensemble des clients est donné par l'union des clients avec livraison et des clients avec cueillette. Une flotte de véhicules identiques avec capacité limitée, basée à un seul dépôt, doit accomplir toutes les livraisons et les cueillettes qui s'avèrent profitables tout en respectant les fenêtres de temps des clients. L'objectif est de minimiser le coût total des tournées, moins les revenus engendrés par les cueillettes réalisées. Trois variantes de ce problème se distinguent entre autres par l'ordre imposé entre les livraisons et les cueillettes sont considérées. Une méthode exacte de génération de colonnes est développée pour ce problème. Des résultats numériques sont présentés pour des instances comportant jusqu'à 100 clients.

Mots clés : Problème de tournées de véhicules, livraisons, cueillettes sélectives, fenêtres de temps, génération de colonnes, plus court chemins avec ressources.

Acknowledgments: This work was partly funded by the Canadian Natural Sciences and Engineering Research Council under grants 157935-07 and 39682-05, by FONDECYT grant 1070741, Instituto Milenio “Complex Engineering Systems”, and by the GSEP program of the Canadian bureau for International Education. This support is gratefully acknowledged.
1 Introduction

The purpose of this paper is to develop a branch-and-price algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows (VRPDSPTW) defined as follows. Let $G = (V, A)$ be a directed graph where $V$ is the vertex set and $A$ is the arc set. The vertex set is partitioned into $V = D \cup P \cup \{0, o^\prime\}$, where $D$ is the set of delivery customers, $P$ is the set of pickup customers, and $o$ and $o^\prime$ represent two copies of the depot called the source and the sink, respectively. Servicing a customer $i \in D \cup P$ takes $s_i$ time units. A time window $[a_i, b_i]$ is imposed on the start of service at customer $i$. Every customer $i \in D$ has a delivery demand $d_i$ and every customer $i \in P$ has a pickup demand $p_i$. All deliveries must be performed, whereas pickups are selective. Performing a pickup at customer $i \in P$ yields a revenue $u_i$. For notational convenience, we define $s_o = s_{o^\prime} = 0$, $d_i = 0$ for all $i \in V \setminus D$, $p_i = u_i = 0$ for all $i \in V \setminus P$, $[a_o, b_o] = [0, 0]$, and $[a_{o^\prime}, b_{o^\prime}] = [0, \bar{b}]$, where $0$ and $\bar{b}$ are constants defining unrestrictive windows at vertices $o$ and $o^\prime$. At the depot, we assume that there is a fleet of identical vehicles of capacity $Q$ sufficient to service all delivery customers. The arc set is given by $A = \{(i, j) : i, j \in V, i \neq j, a_i + s_i + t_{ij} \leq b_j\}$. A travel cost matrix $(c_{ij})$ and a travel time matrix $(t_{ij})$ are defined on $A$. Note that $c_{ij}$ can also include real vehicle fixed costs if any, or very large artificial costs if the number of vehicles used has to be minimized. The VRPDSPTW consists of determining vehicle routes of minimum net cost (travel cost minus revenue) such that 1) all routes start at $o$ and end at $o^\prime$; 2) all deliveries are performed; 3) the vehicle load along its route never exceeds $Q$ and 4) time windows are respected (vehicles are allowed to wait at a customer location before service starts).

The VRPDSPTW belongs to the class of one-to-many-to-one pickup and delivery problems, meaning that all delivery demands originate at the depot and all pickup demands are destined to the depot (Hernández-Pérez and Salazar-González, 2004; Berbeglia et al., 2007). This class of problems includes the single demand case, denoted $P/D$, where delivery and pickup customers are disjoint, and the combined demand case, denoted $P&D$, where the same customer may have a pickup and a delivery. In the latter case, it is possible to create two copies of the same customer (one for the delivery and one for the pickup) and include the copies in $D$ and $P$. By suitably defining the arcs incident to these copies, one can impose that every customer be visited exactly once (the single visit case) or one may let the delivery and pickup operations of a customer to be performed during two separate visits (the multiple visit case). Obviously, in the single visit case, the delivery vertex of a customer should always be visited before its pickup vertex. The single demand case as well as the combined demand and multiple visit case can be subdivided into the mixed case in which pickup and delivery vertices may be visited in any order, and the backhaul case in which all pickup vertices of a given route must be visited after the delivery vertices of the same route. The latter problem is usually called the Vehicle Routing Problem with Backhauls (VRPB). Further details on this classification are provided in Berbeglia et al. (2007) and in Gribkovskaia and Laporte (2008). Our problem statement and algorithm allow the treatment of all variants just described.

One-to-many-to-one pickup and delivery problems arise naturally in reverse logistics operations in which full bottles or containers must be delivered and empty ones are collected (Dethloff, 2001; Tang and Galvão, 2002, 2006; Privé et al., 2006; Hoff et al., 2009). Min (1989) describes an application related to a public library system, whereas Wasner and Zäpfel (2004) consider the case of mail transportation.

For problems without time windows in which all pickups must be performed, an exact branch-and-cut algorithm was described by Baldacci, Hadjiconstantinou and Mingozzi (2003) for the single vehicle case, and a branch-and-price scheme was developed by Dell’Amico, Righini and Salani (2006) for the combined demand case. Toth and Vigo (1997) have developed a branch-and-bound algorithm for the VRPB, whereas Mingozzi, Giorgi and Baldacci (1999) formulate the problem as an integer linear program and solve it exactly by a commercial solver after having reduced the number of variables.

Numerous heuristics have been proposed for one-to-many-to-one pickup and delivery problems. Since the focus of our paper is on the development of an exact algorithm, we refer the reader to the surveys of Toth and Vigo (2002); Berbeglia et al. (2007), and Parragh, Doerner and Hartl (2008) for details about these heuristics.
Relatively little research has been conducted on pickup and delivery problems with selective pickups. Privé et al. (2006) have developed a heuristic for a practical problem involving the delivery of soft-drinks and the collection of empty cans and bottles to and from convenience stores in the Quebec City area. As in our problem, a revenue was associated on selective pickups, and vehicle capacity constraints prevented the collection of all pickups. Gribkovskaia, Laporte and Slyshou (2008) have applied tabu search to the single vehicle pickup and delivery problem with selective pickups. An exact branch-and-bound algorithm was developed by Siral and Bookbinder (2003) for the single vehicle case and a branch-and-cut algorithm was later proposed by Gutiérrez-Jarpa, Obreque and Marianov (2008).

Our aim is to develop an exact branch-and-price algorithm for the three variants of the VRPDSPTW just introduced. With respect to previous contributions, we consider several vehicles, as well as time windows. The algorithm is described in Section 2, followed by computational results in Section 3 and by conclusions in Section 4.

2 Algorithm

Before describing the proposed branch-and-price algorithm, we introduce a generic integer linear program that models all cases of the VRPDSPTW mentioned above. Let $R$ be the set of all feasible vehicle routes. Let $c_r$ be the net cost of route $r$ (computed as the sum of the costs $c_{ij}$ of the arcs in $r$ minus the sum of the revenues $u_i$ of the pickup customers visited), and let $\delta_{ir}$ be a binary parameter equal to 1 if and only if route $r$ visits customer $i$. For each route $r$, define a binary variable $y_r$ equal to 1 if and only if route $r$ is selected.

With this notation, the VRPDSPTW can be formulated as the following integer linear program:

$$\begin{align*}
\text{minimize} & \quad \sum_{r \in R} c_r y_r \\
\text{subject to} & \quad \sum_{r \in R} \delta_{ir} y_r = 1, & \forall i \in D \\
& \quad \sum_{r \in R} \delta_{ir} y_r \leq 1, & \forall i \in P \\
& \quad y_r \in \{0,1\}, & \forall r \in R.
\end{align*}$$

The objective function (1) minimizes the total net cost of the selected routes. Constraints (2) ensure that each delivery customer is visited by exactly one vehicle, while constraints (3) stipulate that each pickup customer can be visited at most once.

In practice, model (1)–(4) contains a very large number of variables, namely, one for each feasible route. To overcome this difficulty, we propose solving it by means of a branch-and-price algorithm that does not require the explicit enumeration of all variables. Branch-and-price (Barnhart et al., 1998; Desaulniers et al., 1998; Desrosiers and Lübbecke, 2005) consists of a column generation algorithm embedded within a branch-and-bound scheme. Column generation is used to compute lower bounds at each node of the branch-and-bound search tree, while branch-and-bound allows the identification of an optimal integer solution.

In Sections 2.1 to 2.3, we provide the details of the branch-and-price algorithm used for the VRPDSPTW for the single demand and mixed case. Section 2.4 discusses the adaptations required to handle combined demands and backhauling.

2.1 Column generation

We now describe the column generation algorithm applied at the root node linear relaxation of model (1)–(4). The adaptation of this algorithm to the other linear relaxations is straightforward. In a column generation context, the linear relaxation is called the master problem.
Column generation is an iterative algorithm which solves a restricted master problem (RMP) and a subproblem at each iteration. At iteration $\ell$, the RMP is simply the master problem restricted to a subset $R_\ell$ of its variables. It is solved by a linear programming solver (we used the primal simplex algorithm) to provide a pair of optimal primal and dual solutions. The subproblem is an elementary shortest path problem with resource constraints (ESPPRC) whose arc costs depend on the RMP dual solution (see the details below). Its goal is to identify variables (columns) in the set $R \setminus R_\ell$ that have a negative reduced cost with respect to this dual solution. If no such variables exist, then the current RMP optimal primal solution is also optimal for the master problem (setting all $y_r$ variables to 0 for $r \in R \setminus R_\ell$) and the algorithm stops. Otherwise, these columns are added to the current RMP before starting a new iteration. The subproblem is solved by the label-setting algorithm described in Section 2.2.

To check whether negative reduced cost variables exist, the subproblem consists of finding a route of least reduced cost. The reduced cost $\tilde{c}_r$ of a route $r \in R$ is given by

$$\tilde{c}_r = c_r - \sum_{i \in D \cup P} \pi^\ell_i \delta_{ir},$$

where $\pi_i$, $i \in D \cup P$, is the dual variable associated with the corresponding constraint (2) or (3), and $\pi^\ell_i$ is its value in the RMP dual solution at iteration $\ell$. Because route $r$ corresponds to a path from $o$ to $o'$ in $G$, its reduced cost can also be written as

$$\tilde{c}_r = \sum_{(i,j) \in A_r} (c_{ij} - u_i - \pi^\ell_i),$$

where $A_r$ is the set of arcs in the corresponding path and $\pi^\ell_0 = 0$. Hence, replacing in $G$ all arc costs $c_{ij}$, $(i, j) \in A$, by arc reduced costs $\tilde{c}_{ij} = c_{ij} - u_i - \pi^\ell_i$, the cost of an $(o, \ldots, o')$ path corresponding to a feasible route $r \in R$ becomes the reduced cost of this route.

However, not all $(o, \ldots, o')$ paths in $G$ correspond to a feasible route. Indeed, the network structure does not enforce path elementarity, time window restrictions, and vehicle capacity. These constraints can be handled by using resources, as described in Irnich and Desaulniers (2005). A resource is a quantity that varies generally along a path, and its value at each visited vertex is restricted to fall within a prespecified resource interval, called a resource window. A resource value exceeding the window upper bound at a vertex indicates that the path is infeasible, while a value that is less than the window lower bound indicates that some of the resource has to be wasted to reach this bound. A typical resource is time which is used to impose the time window constraints. For this resource, the window is $[a_i, b_i]$ at every vertex $i \in V$. The value of this resource is thus equal to 0 at vertex $o$ and increases by at least $t_{ij} + s_i$ on every arc $(i,j)$ contained in a path. If the minimum increase $t_{ij} + s_i$ along an arc $(i,j)$ with $j \in D \cup P$ yields a time value less than $a_j$, then this value is simply raised to $a_j$, meaning that the vehicle has to wait until the beginning of the time window.

For our problem, $3 + |D \cup P|$ resources are needed to restrict the feasibility of every $(o, \ldots, o')$ path in $G$. They are a time resource ($eTime$) to respect the customer time windows; a maximum load resource ($eMaxL$) to enforce vehicle capacity; a total picked-up demand resource ($ePick$) required to compute the value of resource $eMaxL$; and a binary resource ($eCust_k$) for each customer $k \in D \cup P$ to impose path elementarity. The set of resources is denoted by $E$. Note that all these resources have already been introduced in the literature (see Irnich and Desaulniers, 2005).

The resource windows are as follows. For the $eTime$ resource, they are as described above. For the $eMaxL$ and $ePick$ resources, they are equal to $[0, Q]$ for all $i \in V$. For all $eCust_k$ resources, they are equal to $[0, 1]$ for all $i \in V$. The window of a resource $e$ at a vertex $i$ is denoted $[a^e_i, b^e_i]$.

As mentioned above, the value of a resource $e$ varies in general along an $(o, \ldots, o')$ path. In fact, it may vary along each arc contained in this path. The value taken at a vertex $i$ is a variable $T^e_i$ restricted by the window $[a^e_i, b^e_i]$. The vector of the resource values at vertex $i$ is denoted $T_i = (T^e_i)_{e \in E}$. The minimum value that a resource $e$ can take at vertex $j$ when arc $(i, j)$ is used to reach this vertex is given by a function $f^e_{ij}(T_i)$,
called a resource extension function, which can depend on all resource values at vertex \( i \). These functions are as follows:

\[
\begin{align*}
    f_{ij}^{\text{Time}}(T_i) &= T_i^{\text{Time}} + t_{ij} + s_i \\
    f_{ij}^{\text{MaxL}}(T_i) &= \max\{T_i^{\text{MaxL}} + d_j, T_i^{\text{Pick}} + p_j\} \\
    f_{ij}^{\text{Pick}}(T_i) &= T_i^{\text{Pick}} + p_j \\
    f_{ij}^{\text{Custk}}(T_i) &= \begin{cases} 
        T_i^{\text{Custk}} + 1 & \text{if } k = j \\
        T_i^{\text{Custk}} & \text{otherwise}
    \end{cases} \quad \forall k \in D \cup P,
\end{align*}
\]

Therefore, if arc \((i, j)\) is used in a path, then \( T_j^{e} \geq f_{ij}^{e}(T_i), \forall e \in E \). Because there is no reason for wasting any of the last \( 2 + |D \cup P| \) resources, this inequality can be replaced by an equality for these resources. The interpretation of these extension functions is straightforward except for the \( \text{MaxL} \) resource which can be explained as follows. If \( j \in D \) (that is, \( d_j > 0 \) and \( p_j = 0 \)), then the maximum load along the path \((o, \ldots, j)\) increases by \( d_j \) because this additional delivered demand was loaded in the vehicle at the depot \( o \). If \( i \in P \) (that is, \( d_j = 0 \) and \( p_j > 0 \)), then the maximum load along the \((o, \ldots, j)\) path was either reached prior to vertex \( j \) or is reached at this vertex with the loading of this additional picked up demand. Finally, if \( j = o' \) (that is, \( d_j = p_j = 0 \)), then the resource value does not change.

To formulate the ESPPRC subproblem for the VRPDSPTW, we define a binary arc flow variable \( x_{ij} \) for each arc \((i, j) \in A\) that indicates whether or not arc \((i, j)\) is part of the optimal path. Using this notation, the ESPPRC subproblem can be formulated as the following integer nonlinear program:

\[
\begin{align*}
    \text{minimize} & \quad \sum_{(i,j) \in A} \bar{c}_{ij} x_{ij} \\
    \text{subject to} & \quad \sum_{j: (i,j) \in A} x_{ij} = 1, \quad \forall i \in D \cup P \\
    & \quad \sum_{j: (i,j) \in A} x_{ij} - \sum_{j: (j,i) \in A} x_{ji} = 0, \quad \forall i \in D \cup P \\
    & \quad \sum_{j: (i,j') \in A} x_{ij'} = 1, \quad \forall (i,j') \in A \\
    & \quad x_{ij}(f_{ij}^{\text{Time}}(T_i) - T_j^{\text{Time}}) \leq 0, \quad \forall (i,j) \in A \tag{11} \\
    & \quad x_{ij}(f_{ij}^{e}(T_i) - T_j^{e}) = 0, \quad \forall (i,j) \in A, e \in E \setminus \{\text{Time}\} \tag{12} \\
    & \quad a_i^e \leq T_i^{e} \leq b_i^e, \quad \forall i \in V, e \in E \tag{13} \\
    & \quad x_{ij} \in \{0, 1\}, \quad \forall (i,j) \in A. \tag{14}
\end{align*}
\]

In this model, the objective function (7) minimizes the sum of the arc reduced costs. Constraints (8)–(10) define the structure of path \((o, \ldots, o')\). Constraints (11)–(12) impose (minimum) resource values at vertex \( j \) when an arc \((i, j)\) is part of the path, while constraints (13) enforce the resource windows. The ESPPRC subproblem is not solved as an integer nonlinear program, but by means of the label-setting algorithm described next.

### 2.2 A label-setting algorithm

To solve the ESPPRC subproblem, we apply a sophisticated label-setting algorithm similar to the one used by Desaulniers, Lessard and Hadjar (2008), and incorporating different acceleration techniques. We first present the main concepts behind a basic (mono-directional search) label-setting algorithm, and then we discuss the acceleration techniques.

A label-setting algorithm relies on multi-dimensional labels to represent (partial) paths from the source \( o \) to any vertex \( i \in V \). For the VRPDSPTW, such a label \( L_i = (Z_i, T_i^{\text{Time}}, T_i^{\text{MaxL}}, T_i^{\text{Pick}}, (T_i^{\text{Cust}})_k \in D \cup P) \)
representing a partial path ending at vertex $i$ contains a component $Z_i$ indicating the (reduced) cost of this path, and a component $T^*_{ij}$ for each resource $e \in E$ providing the resource values at vertex $i$. Propagating an initial label $L_o = (0, 0, 0, (0)_{k \in D \cup P})$ associated with vertex $o$, a mono-directional search algorithm creates new labels using extension functions. A label $L_i$ is extended along all arcs $(i, j) \in A$ to create new labels:

$$L_j = \left( Z_i + c_{ij}, \max\{a_{ij}^{\text{Time}}, f_{ij}^{\text{Time}}(T_i)\}, j_{ij}^{\text{MaxL}}(T_i), j_{ij}^{\text{Pick}}(T_i), (j_{ij}^{\text{Custk}}(T_i))_{k \in D \cup P} \right).$$

A label $L_j$ is discarded if at least one of its resource components $T^*_j$, $e \in E$, exceeds $b^*_j$, the corresponding resource window upper bound.

To avoid enumerating all feasible paths, a dominance rule is applied to eliminate labels that are not Pareto-optimal and, therefore, cannot yield an optimal path. Given two labels $L_1 = (Z_1, T_1^{\text{Time}}, T_1^{\text{MaxL}}, T_1^{\text{Pick}}, (T_1^{\text{Custk}})_{k \in D \cup P})$, $\ell = 1, 2$, representing partial paths ending at the same vertex, this rule stipulates that $L_1$ dominates $L_2$ (which, in this case, can be discarded) if $L_1 \leq L_2$ (component-wise) and the inequality is strict for at least one component. When all components are equal, then one of the two labels must be kept while the other can be discarded. Further details about basic label-setting algorithms can be found in Irnich and Desaulniers (2005).

To speed up the solution process, we use three different acceleration techniques. The first technique, developed by Righini and Salani (2006), is called \textit{bounded bi-directional search}. Instead of only extending labels forwardly from the source $o$ to the other vertices, it also extends labels backwardly from the sink $o'$. This technique proceeds in three steps. First, as in the mono-directional search algorithm, labels are extended forwardly from $o$ without creating labels having a time component $T^{\text{Time}}$ greater than $H = 0.5 \bar{h}$, the halfway point of the planning horizon. Second, labels are extended backwardly from $o'$ using backward extension functions. A backward label is extended only if its time component exceeds $H$. Third, pairs of forward and backward labels associated with the same vertex are joined together to obtain complete paths, and infeasible paths are rejected.

As suggested by Righini and Salani (2006) for the variant of the VRP-DPSPTW without time windows and with mandatory pickups, the $e$\text{Pick}$ resource must be replaced in the backward step by a resource $e$\text{Del}$ that computes the total delivered demand. Furthermore, the $e$\text{MaxL}$ resource is computed as if the vehicle was performing the route backwards from the sink to the source, picking up every delivered demand and delivering every picked up demand. Denote by $\lambda_j = \left( \zeta_j, \theta^j_{\text{Time}}, \theta^j_{\text{MaxL}}, \theta^j_{\text{Del}}, (\theta^j_{\text{Custk}})_{k \in D \cup P} \right)$ a backward label associated with a partial path $(j, \ldots, o')$. The backward step starts with an initial label $\lambda_o = (0, \overline{0}, 0, 0, (0)_{k \in D \cup P})$ and extends backwardly every non-dominated label $\lambda_j$ at every vertex $j$ such that $\theta^j_{\text{Time}} > H$ along all arcs $(i, j) \in A$ using the following backward resource extension functions:

$$
g^\text{Time}_{ij}(\theta_j) = \theta^j_{\text{Time}} - t_{ij} - s_i^n
$$

$$
g^\text{MaxL}_{ij}(\theta_j) = \max\{g^\text{MaxL}_{ij}(\theta_j) + p_i, \theta^j_{\text{Del}} + d_i \}
$$

$$
g^\text{Del}_{ij}(\theta_j) = \theta^j_{\text{Del}} + d_i
$$

$$
g^\text{Custk}_{ij}(\theta_j) = \begin{cases} 
\theta^j_{\text{Custk}} + 1 & \text{if } k = i \\
\theta^j_{\text{Custk}} & \text{otherwise} 
\end{cases} \forall k \in D \cup P,
$$

where $\theta_j = (\theta^j_e)_{e \in E}$ is the vector of the resource values of label $\lambda_j$. The label created at vertex $i$ is

$$\lambda_i = \left( \zeta_i + c_{ij}, \min\{g^\text{Time}_{ij}(\theta_j), g^\text{MaxL}_{ij}(\theta_j), g^\text{Del}_{ij}(\theta_j), (g^\text{Custk}_{ij}(\theta_j))_{k \in D \cup P} \right).$$

This label is discarded if $T^{\text{Time}}_{ij} = \min\{b^\text{Time}_{ij}, g^\text{Time}_{ij}(\theta_j)\} < \lambda_i^{\text{MaxL}}$ or if at least one of its other resource components $T^*_j$, $e \in E \setminus \{e\text{Time}\}$, exceeds $b^*_j$. In the backward step, the dominance rule must also be adapted as follows. Let $\lambda_\ell = \left( \zeta_\ell, \theta^\ell_{\text{Time}}, \theta^\ell_{\text{MaxL}}, \theta^\ell_{\text{Del}}, (\theta^\ell_{\text{Custk}})_{k \in D \cup P} \right)$, $\ell = 1, 2$, be two labels representing partial paths starting at the same vertex. Then $\lambda_1$ dominates $\lambda_2$ if $\lambda_1 \leq \lambda_2$ for all components except the $e$\text{Time}$ component and $\theta^2_{\text{Time}} \leq \theta^1_{\text{Time}}$ (the inequality must be strict for at least one component).
In the third step of the bounded bi-directional search algorithm, labels are joined together to form complete \((o, \ldots, o')\) paths. Path feasibility must, however, be checked as follows. Let \(L_i = \left(Z_i, T_{i}^{cTime}, T_{i}^{cMaxL}, T_{i}^{cPick}, \right)\) and \(eCust_k = (\lambda_i, \theta_i^{cTime}, \theta_i^{cMaxL}, \theta_i^{cDel}, \theta_i^{cCustk})\) be a pair of forward and backward labels associated with \(i \in D \cup P\). Concatenating the \((o, \ldots, i)\) path represented by \(L_i\) with the \((i, \ldots, o')\) path represented by \(eCust_k\) yields a path \((o, \ldots, o')\) of reduced cost \(Z_i + \lambda_i\). This path is feasible if all the following conditions hold:

\[
\begin{align*}
T_{i}^{cTime} &\leq \theta_i^{cTime} \\
T_{i}^{cMaxL} + \theta_i^{cDel} &\leq Q \\
T_{i}^{cPick} + \theta_i^{cMaxL} &\leq Q \\
T_{i}^{cCustk} + \theta_i^{cCustk} &\leq 1, \quad \forall k \in (D \cup P) \setminus \{i\}.
\end{align*}
\]

Observe that the left-hand side of the second (resp. third) condition provides the maximum load in the vehicle on the \((o, \ldots, i)\) (resp. \((i, \ldots, o')\)) portion of the path. Therefore, together these two conditions ensure that the vehicle capacity is not exceeded all along the path. The interpretation of the first and last conditions are straightforward.

The second acceleration technique, which was developed independently by Boland et al. (2006) and Righini and Salani (2008), is called decremental search space. It starts by solving the subproblem without considering any elemental requirements, that is, without any customer resources \(eCust_k\), \(k \in D \cup P\). If the computed shortest path is non-elementary, the resource \(eCust_k\) of a customer \(k\) visited more than once on this path is added to the subproblem to forbid this path, and the subproblem is solved again. This iterative process is repeated until an elementary shortest path is found. In a column generation context where only negative reduced cost paths are sought, it can however be halted before optimality when either negative reduced cost elementary paths are found, or the length of the shortest path computed at an iteration is non-negative. As proposed by Desaulniers, Lessard and Hadjar (2008), instead of starting this procedure with an empty set of customer resources at each column generation iteration, we start it using the customer resources considered at the end of the previous iteration.

The third technique, which was proposed by Feillet et al. (2004), consists of redefining the \(eCust_k\) resources to increase the number of dominated labels. Instead of setting the value of this resource to 1 only when customer \(k\) has been visited along a path, it also sets it to 1 when the customer cannot be visited anymore because of the time window constraints.

### 2.3 Branch-and-bound

To obtain integer solutions, the column generation algorithm is embedded into a branch-and-bound search tree, which is explored using a best-first strategy. As explained in the literature on branch-and-price (see, e.g., Desaulniers et al., 1998), branching decisions can hardly be made directly on the master problem binary variables \(y_r\), \(r \in R\). Indeed, fixing such variables at 1 can be done easily by modifying the master problem. However, fixing them at 0 requires forbidding the subproblem from generating the corresponding route again, which significantly complexifies the solution process. Consequently, as proposed by several authors, we branch on binary arc flow variables \(w_{ij}\), \((i, j) \in A\), which can be computed as

\[
w_{ij} = \sum_{r \in R} \alpha_{ijr} y_r,
\]

where \(\alpha_{ijr}\) is a binary parameter equal to 1 if and only if route \(r\) contains arc \((i, j)\)...

To fix a variable \(w_{ij}\) at 0, we delete \((i, j)\) from the arc set \(A\) and remove from the master problem all variables \(y_r\) such that route \(r\) contains arc \((i, j)\). To fix \(w_{ij}\) at 1, we delete from \(A\) all arcs \((i, j')\) and \((i', j)\) such that \(j' \neq j\) and \(i' \neq i\), and remove from the master problem all variables \(y_r\) such that route \(r\) contains at least one of these deleted arcs.
2.4 Adaptations for the combined demand and backhaul cases

The different VRPDSPTW variants stated in the introduction can all be solved using the branch-and-price algorithm described above after suitably modifying \( G \) to bring the problem into the same form as the single demand case (P/D). Below, \( \tilde{V} \) and \( \tilde{A} \) denote the modified vertex and arc sets, respectively. The required modifications for each case are as follows.

In the combined demand (P&D) and single visit case, each customer \( i \) can have a pickup and a delivery demand. Then, \( \tilde{V} \) contains two nodes \( i_P \) and \( i_D \) for each customer \( i \), where \( i_P \in P \) and \( i_D \in D \) are its pickup vertex and its delivery vertex, respectively. Every arc \( (j, i) \in A \) is replaced in \( \tilde{A} \) by an arc \( (o, i_P) \) if \( j = o \), or by an arc \( (i_P, j_D) \) otherwise, while every arc \( (i, j) \in A \) is replaced by two arcs \( (i_D, o') \) and \((i_P, o')\) if \( j = o' \), or by two arcs \( (i_D, j_D) \) and \((i_P, j_D)\) otherwise. The arc set \( \tilde{A} \) also contains an arc \((i_D, i_P)\) for each customer \( i \) with \( c_{i_D i_P} = t_{i_D i_P} = 0 \). This arc is the only arc into \( i_P \), ensuring a single visit to \( i \) whenever both the delivery and the pickup are performed. Note that a single \( eCust_i \) resource is needed for each customer \( i \). The value of this resource only increases when vertex \( i_D \) is visited.

The combined demand (P&D) and multiple visit case requires the above modifications and additional arcs. Indeed, for every customer \( i \) and every arc \( (j, i) \in A \), \( \tilde{A} \) contains an arc \((o, i_P)\) if \( j = o \), or two arcs \((j_D, i_P)\) and \((j_P, i_P)\) otherwise. These additional arcs give the possibility of executing a pickup without performing the corresponding delivery immediately before. This case requires, however, two resources per customer (one for the delivery vertex and one for the pickup vertex) to ensure path elementarity, yielding a more difficult ESPPRC subproblem.

In the backhaul case, all deliveries in a route must be performed before any pickup. This case can easily be handled as a single demand problem (P/D) by removing from the arc set \( A \) (or the arc set \( \tilde{A} \) if combined demands and multiple visits are also considered) all arcs \((i, j)\) such that \( i \in P \) and \( j \in D \).

3 Computational results

The branch-and-price algorithm just described was coded in C and tests were run on a Dual Core AMD Opteron processor of 2.6 GHz, using a customized version of the GENCOL software (version 4.5), which is an implementation of a branch-and-price algorithm commercialized by Kronos, Inc. The GENCOL software uses the ILOG CPLEX solver (version 10.1) for the restricted master problems. A time limit of one hour was allowed for the solution of any instance.

Our test problems were derived from the instances created by Solomon (1987). These 100-customer instances are divided into three classes that differ by the geographical distribution of the customers: customers are clustered in the \( C \) instances, randomly distributed in the \( R \) instances and partly clustered, partly randomly distributed in the \( RC \) instances. Smaller instances were generated considering only the first 25 or the first 50 customers. Each instance is represented by a complete graph, and all distances are Euclidean distances calculated with one decimal point and truncations.

We have generated three kinds of problems, all with time windows and selective pickups: single demand problems (P/D) with mixed demands; combined demand problems (P&D) with multiple visits allowed, and single demand problems with backhauls.

In the mixed P/D instances, the pickup customers were designated in function of the size of the instance: for instances with 25 vertices, every vertex with an index equal to a multiple of 3 was designated a pickup customer; for instances with 50 vertices, vertices with an index equal to multiple of 4 were pickup customers; for instances with 100 vertices, vertices with an index equal to a multiple of 5 were pickup customers.

In the P&D problem we generated instances with 25 vertices with a delivery demand equal to the original demand in the Solomon (1987) instance. Pickup demands were defined as follows: starting from vertex 1, the first vertex had a pickup demand equal to the original demand. Vertex 2 had demand equal to 70% of the original demand, and vertex 3 had a demand equal to 130% of the original demand. This sequence was
repeated for all vertices. We considered the same time windows and service times as in the original instances, vertex revenues equal to the value of the pickup demand, and vehicle capacities equal to 50%, 75% and 100% of those of the Solomon instances.

The P/D instances with backhauls are similar to the mixed P/D instances. We only discarded the arcs connecting collection vertices to distribution vertices. For this problem we generated instances with 25 vertices.

Tables 1 to 3 show the effect on resolution difficulty of various parameters such as instance size, vehicle capacity, revenue for pickup vertices, problem type. Each of these three tables is related to the single demand P/D with mixed demands. The first column gives the instance class (for example: C100 refers to “C” instances 101 to 109) and the number of instances in the class. Each of the next three groups of five columns gives results for different values of the parameters. Within each group of four columns, the first column indicates the number of instances for which a proven optimal solution was identified; the second column gives the mean number of pickup vertices served; the column B&B nodes indicates the mean number of nodes in the branch-and-bound tree; the last column is the mean CPU time in seconds. All averages are computed over the number of instances for which a proven optimum was identified.

In Table 1, we solve instances with 25 vertices. The revenue was equal to that of the pickup demand for each served pickup customer. In this table, we compare three capacities: 50, 75 and 100 percent of the vehicle capacity of the original Solomon instances... We observe that instances with a larger vehicle capacity tend to be easier.

In Table 2, we again solve instances with 25 vertices, but with vehicles having a capacity equal to 50% of that of the original instances. We let the revenue be equal to 0.5, 1 and 1.5 times the pickup demand of each pickup customer. When the revenue is smaller, the mean number of pickup vertices served is also smaller, because it is probably not attractive to visit some pickup vertices. There is also less branching when the revenue is smaller.

In Table 3, we consider a vehicle capacity of 50% of that of the original Solomon instances, a revenue equal to the demand of each pickup customer, and different instance sizes (25 vertices, 50 vertices and 100 vertices). Clearly, if the instances are larger they are harder to solve. This can be seen by the number of instances where it is possible to find an optimal solution and the mean time necessary to solve the problem.

Table 4 compares the three variants under study: single demand problems (P/D) with mixed demands, combined demand problems (P&D) with multiple visits allowed, and single demand problems (P/D) with backhauls. We used a capacity equal to 50% of that of the original instances, and a revenue equal to the pickup demand. The most difficult problem to solve is the combined demand case, and the least difficult is the single demand problem with backhauls.

Finally, we observe that in all cases more pickup customers are serviced in clustered instances because the marginal cost of an additional visit in a cluster tends to be small. This geographical effect is not observed in random instances.

4 Conclusions

We have described an exact branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows. The algorithm is capable of solving three versions of the problem: single demands (mixed case), combined demands, and single demands (with backhauls). Instances involving up to 50 customers can be solved optimally. The combined demand case appears to be the most difficult.
Table 1: Single demand problems (P/D), mixed case, with different vehicle capacities with respect to that of the Solomon instances, and revenue equal to pickup demand for instances with 25 vertices, $|D| = 16$ and $|P| = 9$

| Instance class | Number of instances | Capacity 50% |  | Capacity 75% |  | Capacity 100% |  |
|----------------|---------------------|--------------|----------------|--------------|----------------|----------------|
|                |                     | Optimal solutions | Pickup vertices served | B&B nodes | Time (sec) | Optimal solutions | Pickup vertices served | B&B nodes | Time (sec) | Optimal solutions | Pickup vertices served | B&B nodes | Time (sec) |
| C100           | 9                   | 7             | 9             | 1400        | 449.2        | 9             | 1             | 3.8          | 9             | 9             | 0             | 3.0           |
| C200           | 8                   | 7             | 9             | 4           | 111.1        | 7             | 9             | 4            | 108.9         | 7             | 9             | 0             | 84.9          |
| R100           | 12                  | 12            | 6             | 8           | 0.8          | 12            | 6             | 8            | 0.8          | 12            | 6             | 0             | 1.5           |
| R200           | 11                  | 11            | 6             | 5           | 267.4        | 10            | 6             | 6            | 164.9         | 10            | 6             | 0             | 259.4         |
| RC100          | 8                   | 8             | 8             | 330         | 48.1         | 8             | 9             | 2            | 0.8          | 8             | 9             | 0             | 0.7           |
| RC200          | 8                   | 5             | 9             | 1           | 150.0        | 5             | 9             | 1            | 150.3         | 6             | 9             | 0             | 302.6         |

Table 2: Single demand problems (P/D), mixed case, for different revenues and vehicle capacity equal to 50% of that of the Solomon instances, for instances with 25 vertices, $|D| = 16$ and $|P| = 9$

<table>
<thead>
<tr>
<th>Instance class</th>
<th>Number of instances</th>
<th>Revenue = 0.5 pickup demand</th>
<th></th>
<th>Revenue = pickup demand</th>
<th></th>
<th>Revenue = 1.5 pickup demand</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Optimal solutions</td>
<td>Pickup vertices served</td>
<td>B&amp;B nodes</td>
<td>Time (sec)</td>
<td>Optimal solutions</td>
<td>Pickup vertices served</td>
</tr>
<tr>
<td>C100</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>714</td>
<td>116.7</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>C200</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>209.2</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>R100</td>
<td>12</td>
<td>12</td>
<td>4</td>
<td>6</td>
<td>0.4</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>R200</td>
<td>11</td>
<td>11</td>
<td>4</td>
<td>4</td>
<td>97.5</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>RC100</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>316</td>
<td>43.2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>RC200</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>1</td>
<td>95.6</td>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>
Table 3: Single demand problems (P/D), mixed case, for different sizes, revenue equal to the pickup demand and 50% vehicle capacity with respect to that of the Solomon instances

<table>
<thead>
<tr>
<th>Instance class</th>
<th>Number of instances</th>
<th>25 vertices</th>
<th>50 vertices</th>
<th>100 vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>D = 16 and P = 9</td>
<td>D = 37 and P = 13</td>
<td>D = 80 and P = 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimal solutions</td>
<td>Pickup vertices served</td>
<td>B&amp;B nodes</td>
</tr>
<tr>
<td>C100</td>
<td>9</td>
<td>9</td>
<td>1400</td>
<td>492.4</td>
</tr>
<tr>
<td>C200</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>R100</td>
<td>12</td>
<td>12</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>R200</td>
<td>11</td>
<td>11</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>RC100</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>330</td>
</tr>
<tr>
<td>RC200</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4: Different problems with fixed deliveries and selective pickups, revenue equal to the pickup demand and capacity equal to 50% of that of the Solomon instances

<table>
<thead>
<tr>
<th>Instance class</th>
<th>Number of instances</th>
<th>Single demand (P/D) mixed</th>
<th>D = 16 and P = 9</th>
<th>Combined demand (P&amp;D)</th>
<th>D = 25 and P = 25</th>
<th>Single demand (P/D) with backhauls</th>
<th>D = 16 and P = 9</th>
<th>B&amp;B nodes</th>
<th>Time (sec)</th>
<th>Optimal solutions</th>
<th>Pickup vertices served</th>
<th>B&amp;B nodes</th>
<th>Time (sec)</th>
<th>Optimal solutions</th>
<th>Pickup vertices served</th>
<th>B&amp;B nodes</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C100</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>1400</td>
<td>449.2</td>
<td>3</td>
<td>21</td>
<td>4285</td>
<td>1364</td>
<td>9</td>
<td>6</td>
<td>0</td>
<td>519.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C200</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>4</td>
<td>111.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>4.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R100</td>
<td>12</td>
<td>12</td>
<td>8</td>
<td>0.8</td>
<td>12</td>
<td>22</td>
<td>6</td>
<td>12.6</td>
<td>12</td>
<td>4</td>
<td>0</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R200</td>
<td>11</td>
<td>11</td>
<td>6</td>
<td>5</td>
<td>267.4</td>
<td>1</td>
<td>23</td>
<td>3</td>
<td>29.6</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>34.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC100</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>330</td>
<td>48.1</td>
<td>8</td>
<td>25</td>
<td>41</td>
<td>160.1</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>10.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC200</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>150.0</td>
<td>1</td>
<td>25</td>
<td>1</td>
<td>624.9</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>28.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
References


