CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

Branch-and-Cut-and-Price for the Pickup and Delivery Problem with Time Windows

Stefan Røpke
Jean-François Cordeau

July 2008

CIRRELT-2008-33

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Université Laval · UQÀM · HEC Montréal · École Polytechnique Montréal · Université de Montréal

Branch-and-Cut-and-Price for the Pickup and Delivery Problem with Time Windows†

Stefan Røpke[1,2], Jean-François Cordeau[1,3,*]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] DIKU, University of Copenhagen, Universitetsparken 1, 2100 Copenhagen, Denmark

[3] Canada Research Chair in Logistics and Transportation, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Abstract. In the pickup and delivery problem with time windows (PDPTW), vehicle routes must be designed to satisfy a set of transportation requests, each involving a pickup and a delivery location, under capacity, time window, and precedence constraints. This paper introduces a new branch-and-cut-and-price algorithm in which lower bounds are computed by solving through column generation the linear programming relaxation of a set partitioning formulation. Two pricing subproblems are considered in the column generation algorithm: an elementary and a non-elementary shortest path problem. Valid inequalities are added dynamically to strengthen the relaxations. Some of the previously proposed inequalities for the PDPTW are also shown to be implied by the set partitioning formulation. Computational experiments indicate that the proposed algorithm outperforms a recent branch-and-cut algorithm.

Keywords. Pickup and delivery, column generation, branch-and-price, branch-and-cut, valid inequalities.

Acknowledgements. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant 227837-04. This support is gratefully acknowledged. We are also grateful to David Pisinger and Stefan Irnich for their valuable comments and suggestions.

†Revised version of the CRT-2006-21

---

* Corresponding author: Jean-Francois.Cordeau@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2008

1 Introduction

In the classical Vehicle Routing Problem (VRP), a fleet of vehicles based at a common depot must be routed to visit exactly once a set of customers with known demand. Each vehicle route must start and finish at the depot and the total demand of the customers visited by the route must not exceed the vehicle capacity. In the VRP with Time Windows (VRPTW), a time window is associated with each customer and the vehicle visiting a given customer cannot arrive after the end of the time window. The Pickup and Delivery Problem with Time Windows (PDPTW) is a further generalization of the VRP in which each customer request is associated with two locations: an origin location where a certain demand must be picked up and a destination where this demand must be delivered. Each route must also satisfy pairing and precedence constraints: for each request, the origin must precede the destination, and both locations must be visited by the same vehicle. The VRPTW can be seen as a special case of the PDPTW in which all requests have a common origin which corresponds to the depot.

The PDPTW has applications in various contexts such as urban courier services, less-than-truckload transportation, and door-to-door transportation services for the elderly and the disabled. In the latter case, narrow time windows are often considered and ride time constraints are imposed to control the time spent by a passenger in the vehicle. The resulting problem is called the Dial-a-Ride Problem (DARP).

The VRP and VRPTW are well known combinatorial optimization problems which have received a lot of attention (see, e.g., Toth and Vigo, 2002). Since it generalizes the VRPTW, the PDPTW is clearly $\mathcal{NP}$-hard. Over the last few decades, several heuristics have been proposed for the PDPTW. However, because of the difficulty of the problem, work on exact methods has been somewhat limited.

Two main approaches have been used to solve the PDPTW exactly: branch-and-price and branch-and-cut. Branch-and-price methods (see, e.g., Barnhart et al., 1998; Desaulniers et al., 1998) use a branch-and-bound scheme in which lower bounds are computed by column generation. The first branch-and-price algorithm for the PDPTW was proposed by Dumas et al. (1991) who considered a set partitioning formulation of the problem in which each column corresponds to a feasible vehicle route and each constraint is associated to a request that must be satisfied exactly once. The resulting pricing subproblem is a shortest path problem with time window, capacity, pairing and precedence constraints. This problem is solvable by dynamic programming and the authors used an algorithm similar to the one developed by Desrosiers et al. (1986) for the single-vehicle pickup and delivery problem with time windows. Several label elimination methods are proposed to accelerate the dynamic programming algorithm, and arc elimination rules are used to reduce the size of the problem. The authors pointed out that their approach works well when the demand of each customer is large with respect to vehicle capacity. The largest instance solved with their approach contains 55 requests.

Another branch-and-price approach for the PDPTW was later described by Savelsbergh and Sol (1998). Their approach differs from that of Desrosiers et al. in several respects: *i)* whenever possible, they use construction and improvement heuristics to solve the pricing subproblem; *ii)* a sophisticated column management mechanism is used to keep the column generation master problem as small as possible; *iii)* columns are selected with a bias toward

increasing the likelihood of identifying feasible integer solutions during the solution of the master problem; *iv)* branching decisions are made on additional variables representing the fraction of a request that is served by a given vehicle; and *v)* a primal heuristic is used at each node of the search tree to compute upper bounds. Column generation was also used recently by Xu et al. (2003) and Sigurd et al. (2004) to address variants of the PDPTW arising in long-haul transportation planning and in the transportation of live animals, respectively.

The second family of exact approaches for the PDPTW is branch-and-cut. In branch-and-cut, valid inequalities (i.e., cuts) are added to the formulation at each node of the branch-and-bound tree to strengthen the relaxations which are usually solved by the simplex algorithm. Relying on the previous work of Balas et al. (1995) and Ruland and Rodin (1997) on the Precedence-Constrained Traveling Salesman Problem (PCTSP) and the TSP with Pickup and Delivery (TSPPD), Cordeau (2006) developed a branch-and-cut algorithm for the DARP based on a three-index formulation of the problem. This algorithm was able to solve instances with four vehicles and 32 requests. It was later improved by Ropke et al. (2007) who compared different formulations of the DARP and PDPTW, and introduced two new families of inequalities for these problems. One is an adaptation of the *reachability cuts* introduced by Lysgaard (2006) for the VRPTW, while the other is called *fork inequalities*. Both families can also be used in the context of column generation and will be described in Section 4. Using these inequalities, Ropke et al. were able to solve DARP instances with eight vehicles and 96 requests. Another branch-and-cut approach, based on a two-index formulation was proposed by Lu and Dessouky (2004). This formulation contains a polynomial number of constraints, but relies on extra variables to impose pairing and precedence constraints. Instances with up to five vehicles and 25 requests were solved optimally with this approach.

For reviews on pickup and delivery problems, the reader is referred to the works of Savelsbergh and Sol (1995), Desaulniers et al. (2002) and Cordeau et al. (2007).

In this paper, we introduce a new branch-and-cut-and-price algorithm for the PDPTW. It is well known that set partitioning formulations of vehicle routing problems tend to provide stronger lower bounds than formulations based on arc (flow) variables (see Bramel and Simchi-Levi, 2002). Two different shortest path problems have been considered as pricing subproblems for the PDPTW in the literature. In the first application of column generation to the PDPTW (Dumas et al., 1991) a non-elementary shortest path problem was solved while later implementations (Sol, 1994; Sigurd et al., 2004) have used an elementary shortest path problem. Both shortest path problems are $\mathcal{NP}$-hard. Little is known about how the relaxations obtained by solving these two subproblems differ. The only result we are aware of is by Sol (1994) who has shown an example where the objective of the relaxation obtained with the non-elementary problem is half of the objective obtained with the elementary one.

The contributions of this paper are fourfold. First, we present an improved algorithm for the elementary version of the pricing problem. Second, we show how valid inequalities can be introduced in the formulation to improve the quality of the lower bounds. Third, we show that some previously proposed inequalities are implied by the LP relaxation of the set partitioning formulation and that the elementary relaxation implies more inequalities than the non-elementary one. Fourth, we report extensive computational experiments on a large set of instances.

The remainder of the paper is organized as follows. Section 2 defines the PDPTW and introduces mathematical formulations of the problem. Section 3 discusses the two pricing

subproblems that we use within the branch-and-price algorithm. Section 4 describes valid inequalities that can be added to the formulation while Section 5 studies the relationships between these inequalities and the set partitioning formulation of the problem. The branch-and-cut-and-price algorithm is then described in Section 6. Finally, computational results are reported in Section 7, followed by conclusions in the last section.

2 Mathematical Formulation

In this section, we introduce the notation that is used throughout the paper. We then present a standard three-index model of the problem, followed by a set partitioning formulation.

2.1 Notation

Let n denote the number of requests to satisfy. We define the PDPTW on a directed graph $G = (N, A)$ with node set $N = \{0, \ldots, 2n+1\}$ and arc set A. Nodes 0 and $2n+1$ represent the origin and destination depots, while subsets $P = \{1, \ldots, n\}$ and $D = \{n+1, \ldots, 2n\}$ represent pickup and delivery nodes, respectively. Thus, each request i is associated with a pickup node i and a delivery node $n+i$.

With each node $i \in N$ are associated a load q_i and a non-negative service duration d_i satisfying $q_0 = q_{2n+1} = 0$, $q_i = -q_{n+i}$ $(i = 1, \ldots, n)$ and $d_0 = d_{2n+1} = 0$. A time window $[a_i, b_i]$ is also associated with every node $i \in P \cup D$, where a_i and b_i represent the earliest and latest time, respectively, at which service may start at node i. The depot nodes may also have time windows $[a_0, b_0]$ and $[a_{2n+1}, b_{2n+1}]$ representing the earliest and latest times, respectively, at which the vehicles may leave from and return to the depot. Let K denote the set of vehicles. We assume that vehicles are identical and have capacity Q. With each arc $(i, j) \in A$ are associated a routing cost c_{ij} and a travel time t_{ij}. In the remainder of the paper, we assume that the travel time t_{ij} includes the service time d_i at node i. We also assume that the triangle inequality holds both for routing costs and travel times.

2.2 Three-index formulation of the PDPTW

For each arc $(i, j) \in A$ and each vehicle $k \in K$, let x_{ij}^k be a binary variable equal to 1 if and only if vehicle k travels directly from node i to node j. For each node $i \in N$ and each vehicle $k \in K$, let B_i^k be the time at which vehicle k begins service at node i, and Q_i^k be the load of vehicle k upon leaving node i. Using these variables, the PDPTW can be formulated as the following non-linear mixed-integer program:

$$\text{Min} \quad \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}^k \tag{1}$$

subject to

$$\sum_{k \in K} \sum_{j \in N} x_{ij}^k = 1 \qquad \forall i \in P \tag{2}$$

$$\sum_{j \in N} x_{ij}^k - \sum_{j \in N} x_{n+i,j}^k = 0 \qquad \forall i \in P, k \in K \tag{3}$$

$$\sum_{j \in N} x_{0j}^k = 1 \qquad \forall k \in K \tag{4}$$

$$\sum_{j \in N} x_{ji}^k - \sum_{j \in N} x_{ij}^k = 0 \qquad \forall i \in P \cup D, k \in K \tag{5}$$

$$\sum_{i \in N} x_{i,2n+1}^k = 1 \qquad \forall k \in K \tag{6}$$

$$B_j^k \geq (B_i^k + t_{ij}) x_{ij}^k \qquad \forall i \in N, j \in N, k \in K \tag{7}$$

$$Q_j^k \geq (Q_i^k + q_j) x_{ij}^k \qquad \forall i \in N, j \in N, k \in K \tag{8}$$

$$B_i^k + t_{i,n+i} \leq B_{n+i}^k \qquad \forall i \in P \tag{9}$$

$$a_i \leq B_i^k \leq b_i \qquad \forall i \in N, k \in K \tag{10}$$

$$\max\{0, q_i\} \leq Q_i^k \leq \min\{Q, Q + q_i\} \qquad \forall i \in N, k \in K \tag{11}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall i \in N, j \in N, k \in K. \tag{12}$$

The objective function (1) minimizes the total routing cost. Constraints (2) and (3) ensure that each request is served exactly once and that the pickup and delivery nodes are visited by the same vehicle. Constraints (4)-(6) guarantee that the route of each vehicle k starts at the origin depot and ends at the destination depot. Consistency of the time and load variables is ensured by constraints (7) and (8). Constraints (9) ensure that for each request i, the pickup node is visited before the delivery node. Finally, inequalities (10) and (11) impose time windows and capacity constraints, respectively. The model is non-linear because of inequalities (7) and (8) but it can easily be linearized by using standard reformulation techniques.

2.3 Set partitioning formulation of the PDPTW

To formulate the problem as a set partitioning problem, let Ω denote the set of all feasible routes satisfying constraints (3)-(12), dropping index k (as all vehicles are assumed to be identical). For each route $r \in \Omega$, let c_r be the cost of the route and let a_{ir} be a constant indicating the number of times node $i \in P$ is visited by r. Let also y_r be a binary variable equal to 1 if and only if route $r \in \Omega$ is used in the solution. The PDPTW can then be formulated as the following set partitioning problem:

$$\text{Min} \quad \sum_{r \in \Omega} c_r y_r \tag{13}$$

subject to

$$\sum_{r \in \Omega} a_{ir} y_r = 1 \qquad \forall i \in P \tag{14}$$

$$y_r \in \{0, 1\} \qquad \forall r \in \Omega. \tag{15}$$

The objective function (13) minimizes the cost of the chosen routes while constraints (14) ensure that every request is served once. A lower bound on the optimal value of (13)-(15)

can be obtained by solving the linear programming (LP) relaxation which is obtained by replacing the integrality requirements (15) with the simple constraints

$$y_r \geq 0 \qquad \forall r \in \Omega. \tag{16}$$

Because of the large size of set Ω, it is usually very difficult to solve or even to represent model (13)-(15) explicitly. Instead, its LP relaxation is solved using column generation. In a column generation approach, a restricted master problem is obtained by considering a subset $\bar{\Omega} \subseteq \Omega$ of routes. Additional columns of negative reduced cost are generated by solving a *pricing subproblem*. Following Wolsey (1998), we call the problem defined by (13)–(15) the *integer programming master problem* (*IPM*) and its LP relaxation the *linear programming master problem* (*LPM*). The pricing problem for the PDPTW is

$$\text{Min} \quad \sum_{i,j \in N} d_{ij} x_{ij} \tag{17}$$

subject to constraints (3)–(12) (dropping index k), where d_{ij} is defined as

$$d_{ij} = \begin{cases} c_{ij} - \pi_i & \forall i \in P, j \in N \\ c_{ij} & \forall i \in N \setminus P, j \in N, \end{cases} \tag{18}$$

and π_i is the dual variable associated with the set partitioning constraint (14) for node i. We denote this problem as SP1.

The definition of d_{ij} in equation (18) ensures that $d_{ij} + d_{jk} \geq d_{ik}$ if j is a delivery node as c_{ij} satisfies the triangle inequality. We say that a cost matrix that satisfies this property satisfies the *delivery triangle inequality*. As will be shown in Section 3 this is computationally convenient. The problem defined by objective (17) and constraints (3)–(12) is a constrained shortest path problem called the *Elementary Shortest Path Problem with Time Windows, Capacity, and Pickup and Delivery* (ESPPTWCPD). In Section 3 we explain how this and related problems can be solved using label setting algorithms.

Instead of solving the shortest path problem SP1 one can solve relaxed versions of this problem. A relaxed shortest path problem implies that a set of routes Ω' is implicitly considered, where $\Omega \subseteq \Omega'$. If Ω' satisfies the property that none of the columns from the set $\Omega' \setminus \Omega$ can be used in a feasible integer solution to IPM, then the set partitioning problem solved on Ω' will have the same set of optimal solutions as the one solved on Ω. Obviously, the lower bound obtained by solving the LP relaxation on Ω' may, however, be weaker. An example of a relaxation of the elementary shortest path problem that satisfies this property consists of allowing cycles in the path. In this case, some requests may be served more than once. Paths containing cycles cannot, however, appear in a feasible integer solution because of constraints (14). This relaxation was used by Dumas et al. (1991) and it is described in more detail in Section 3.3.

Relaxations inducing sets Ω' for which one cannot ensure that no column from $\Omega' \setminus \Omega$ can belong to a feasible integer solution to IPM can also be used. In this case, however, valid inequalities must be added to the master problem to render such solutions infeasible. This approach was used by Ropke (2005) to solve the PDPTW using more relaxed pricing problems.

Valid inequalities expressed in terms of the x_{ij}^k variables from the three-index formulation (1)-(12) can be added to the master problem following the approach proposed by Kohl et al. (1999) for the VRPTW. Since all vehicles are identical, we first observe that such inequalities can be expressed in terms of x_{ij} variables and can be written in the form

$$\sum_{i=0}^{2n+1} \sum_{j=0}^{2n+1} \alpha_{ij} x_{ij} \geq \beta,$$

where $\alpha_{ij} \in \mathbb{R}$ is the coefficient of arc $(i,j) \in A$ and $\beta \in \mathbb{R}$ is a constant. This inequality is transfered to the master problem as

$$\sum_{r \in \Omega} \phi_r y_r \geq \beta,$$

where $\phi_r = \sum_{(i,j) \in A} \psi_{ijr} \alpha_{ij}$, and ψ_{ijr} is the number of times arc (i,j) is used in route r. The introduction of a valid inequality in the master problem modifies the pricing problem. Indeed, the arc costs d_{ij} are now defined as follows:

$$d_{ij} = \begin{cases} c_{ij} - \pi_i - \alpha_{ij}\mu & \forall i \in P, j \in N \\ c_{ij} - \alpha_{ij}\mu & \forall i \in N \setminus P, j \in N, \end{cases} \tag{19}$$

where μ is the dual variable associated with the added inequality. Any number of inequalities can be added in this way. Notice that unlike definition (18) this new definition of d_{ij} does not satisfy the delivery triangle inequality.

3 Constrained Shortest Path Problems

Resource constrained shortest path problems arising in column generation approaches for vehicle routing problems are typically solved using dynamic programming techniques called labeling algorithms. Notice that the term "shortest path" should be interpreted carefully: given a cost function one wishes to find the least-cost feasible path from the source node to the sink node. An overview of constrained shortest path problems and of appropriate solution techniques is given by Irnich and Desaulniers (2005).

In this section we show how the ESPPTWCPD introduced in Section 2.3 can be solved using a labeling algorithm. A relaxed version of the problem is considered in Section 3.3.

3.1 Label setting shortest path algorithms

Consider a weighted directed graph $G = (N, A)$ where N is the set of nodes, A is the set of arcs, s is the source node and t is the sink node. We assume that no arc enters node s and no arc leaves node t. Let γ be the number of resources in the problem. Traversing arcs consumes resources. Let $f_{ij}^p \in \mathbb{Q}$ denote the consumption of resource $p \in \{1, \ldots, \gamma\}$ for arc $(i,j) \in A$. For every node $i \in N$ lower bounds $l_i^p \in \mathbb{Q}$ and upper bounds $u_i^p \in \mathbb{Q}$ on the resource variables $p \in \{1, \ldots, \gamma\}$ are given.

In label setting shortest path algorithms, a label consists of three elements: a node, the cumulated resource consumption at that node, and a pointer to its parent label. A label

$L = (i, R, p)$ corresponds to a path starting at node s and ending at node i with a certain resource consumption characterized by the vector $R \in \mathbb{Q}^\gamma$. The parent label p is necessary to reconstruct the path between s and i. Resource constrained shortest path problems can be solved using an algorithm based on the pseudo-code presented in Algorithm 1.

Algorithm 1 Pseudo code for labeling algorithm

```
1  Input: graph G = (N, A), source node s, sink node t
2  U = {(s, (l_s^1, ..., l_s^γ), nil)}
3  while U ≠ ∅ do
4    L = removefirst(U)
5    i = node(L)
6    if no label in L_i dominates L then
7      L_i = L_i ∪ {L}
8      extend L along all arcs (i, j) leaving node i
9      add all feasible extensions to U
10 return path corresponding to best label in L_t
```

In line 2 of the algorithm, an initial label $(s, (l_s^1, \ldots, l_s^\gamma), nil)$ corresponding to the source node s is created. In this label, the resource consumption is set according to the lower bounds for node s. Here, U designates the set of unprocessed labels and $\mathcal{L}_i$ is the set of processed labels at node i (paths ending at node i). Lines 4 to 8 are repeated as long as there are unprocessed labels. In line 4 a new unprocessed label is selected using the `removefirst` function (the function removes the label from U). In line 5 the node of the label is retrieved and line 6 checks whether the label can be discarded (this is explained in more detail below). If the label cannot be discarded then it is stored in the set of processed labels for node i in line 7. In line 8, new labels are created by extending label L. Extending a label $L = (i, R, p)$ along arc (i, j) results in the label (j, R', L) where the k^{th} component R'_k of R' is given by either $R'_k = \max\left\{l_j^k, R_k + f_{ij}^k\right\}$ or $R'_k = R_k + f_{ij}^k$ depending on the type of resource. The new label is feasible if all resource variables are within their lower and upper bounds for node j. All labels corresponding to feasible extensions of label L are inserted into U in line 9. In line 10, the label with the least cost at the sink node is returned.

Without the test in line 6 the algorithm is a brute-force approach that enumerates all feasible paths. The test in line 6 removes unpromising labels based on a so called *dominance criterion*. A label L_1 is said to *dominate* label L_2, written $L_1 \preceq_{dom} L_2$ if and only if they are assigned to the same node and no feasible extension of the path corresponding to L_2 with a path to t has a lower cost than the best (with respect to cost) feasible extension of the path corresponding to L_1 with a path to t. If $L_1 \preceq_{dom} L_2$ then there is no need to consider L_2, and we need only examine extensions of L_1.

Given two labels it can be difficult to determine whether one label dominates the other as we potentially have to examine all possible augmentations of the corresponding paths to node t. Consequently we use sufficient (but not necessary) conditions for dominance. In Section 3.2 and 3.3 we describe examples of such conditions.

3.2 ESPPTWCPD - SP1

The ESPPTWCPD, denoted SP1, is the natural pricing problem for the PDPTW and the one that provides the best lower bounds. In the context of the PDPTW, it was first used by Sol (1994) and later by Sigurd et al. (2004) for a PDPTW with additional precedence constraints. Sigurd et al. (2004) have described a general labeling algorithm for the ESPPTWCPD and a more efficient one that takes advantage of the additional precedence constraints. In this section we present a new labeling algorithm for the ESPPTWCPD which contains a less restrictive, sufficient dominance condition compared to the algorithm proposed by Sol (1994) and the general one described by Sigurd et al. (2004). A less restrictive dominance condition implies that more labels can be eliminated, resulting in a better performance of the shortest path algorithm. In what follows we assume that the source and sink nodes are, respectively, 0 and $2n + 1$.

3.2.1 Label management

For each label we store the following data: η – the node of the label, t – the arrival time at the node, l – the load of the vehicle after visiting node η, c – the accumulated cost, $\mathcal{V} \subseteq \{1, \ldots, n\}$ – the set of requests that have been started on the path (and possibly been completed), $\mathcal{O} \subseteq \{1, \ldots, n\}$ – the set of requests that have been started but not completed, i.e., the pickup has been served but not the delivery. The requests in $\mathcal{O}$ are said to be *open*. We also store a pointer to the parent label in each label. Our resources are t, l, c, $\mathcal{V}$ and $\mathcal{O}$. The notation $t(L)$ is used to refer to the arrival time in label L and similar notation is used for the rest of the resources (e.g., $\eta(L)$, $l(L)$, $c(L)$, $\mathcal{V}(L)$ and $\mathcal{O}(L)$).

When extending a label L along an arc $(\eta(L), j)$, the extension is legal only if

$$t(L) + t_{\eta(L),j} \leq b_j \tag{20}$$
$$l(L) + q_j \leq Q. \tag{21}$$

Inequality (20) ensures time window feasibility while inequality (21) ensures capacity feasibility. Furthermore, L and j must satisfy one of the following three conditions:

$$0 < j \leq n \quad \wedge \quad j \notin \mathcal{V}(L) \tag{22}$$
$$n < j \leq 2n \quad \wedge \quad j - n \in \mathcal{O}(L) \tag{23}$$
$$j = 2n + 1 \quad \wedge \quad \mathcal{O}(L) = \emptyset. \tag{24}$$

Condition (22) states that if j is a pickup node then the node has not been visited before on the path. Condition (23) states that if j is a delivery node then the path has already visited the corresponding pickup node, i.e., the precedence relationship between pickups and deliveries is satisfied. Finally, condition (24) states that if j is the sink node then all requests that have been started have also been completed. This condition enforces the pairing constraint: the pickup and delivery from any given request must be served on the same path. In the presence of (23), condition (22) is sufficient to ensure that only elementary paths are considered.

If extension along the arc $(\eta(L), j)$ is feasible then a new label L' is created at node j. The information in label L' is set as follows:

$$\eta(L') = j \tag{25}$$
$$t(L') = \max\{a_j, t(L) + t_{\eta(L),j}\} \tag{26}$$
$$l(L') = l(L) + q_j \tag{27}$$
$$c(L') = c(L) + d_{\eta(L),j} \tag{28}$$
$$\mathcal{V}(L') = \begin{cases} \mathcal{V}(L) \cup \{j\} & \text{if } j \in P \\ \mathcal{V}(L) & \text{if } j \in D \end{cases} \tag{29}$$
$$\mathcal{O}(L') = \begin{cases} \mathcal{O}(L) \cup \{j\} & \text{if } j \in P \\ \mathcal{O}(L) \setminus \{j-n\} & \text{if } j \in D. \end{cases} \tag{30}$$

Equations (25)-(28) set the current node, the time, the load and the cost of the new label, respectively. Equation (29) updates the set of visited requests. Node j is only added if it is a pickup node. Equation (30) updates the set of open requests. If a pickup (resp. delivery) node is visited, the corresponding request is added to (resp. removed from) the set to indicate that the request has been started (resp. completed).

3.2.2 Dominance criterion

The dominance criterion employed in this section, denoted by (DOM1), is the following: a label L_1 dominates a label L_2 if

$$\eta(L_1) = \eta(L_2),\ t(L_1) \leq t(L_2),\ c(L_1) \leq c(L_2),\ \mathcal{V}(L_1) \subseteq \mathcal{V}(L_2),\ \mathcal{O}(L_1) \subseteq \mathcal{O}(L_2). \tag{31}$$

Let $\mathcal{P}(L)$ represent the path corresponding to label L and (p_1, p_2) the path obtained by concatenating paths p_1 and p_2.

Proposition 1. DOM1 is a valid dominance criterion when d_{ij} satisfies the delivery triangle inequality.

Proof. The proof follows from that of Proposition 4 in Dumas et al. (1991). Let p be a feasible path extending $\mathcal{P}(L_2)$ to $2n+1$. If such a path does not exist then clearly one can remove label L_2. Let p' be the path obtained from p by removing the deliveries corresponding to the requests in $\mathcal{O}(L_2) \setminus \mathcal{O}(L_1)$. As $(\mathcal{P}(L_2), p)$ is feasible, then so is $(\mathcal{P}(L_1), p')$. Indeed, it is easy to see that it is feasible with respect to time windows because travel times satisfy the triangle inequality. The capacity is not violated on $(\mathcal{P}(L_1), p')$ as it was not violated on $(\mathcal{P}(L_2), p)$ and $\mathcal{P}(L_1)$ does not visit the pickups corresponding to the deliveries removed from p. It is also easy to see that $(\mathcal{P}(L_1), p')$ is elementary and satisfies pairing constraints. The cost of $(\mathcal{P}(L_1), p')$ is smaller than or equal to the cost of $(\mathcal{P}(L_2), p)$ because $c(L_1) \leq c(L_2)$ and the cost of p' is less than or equal to the cost of p due to the delivery triangle inequality assumption. As a result, the best (with respect to cost) extension of label L_1 to $2n+1$ will always be no worse than the best extension of L_2 to $2n+1$. Hence, label L_1 dominates label L_2. □

Notice that it is not necessary to consider the load of a label in the dominance criterion: since $\mathcal{O}(L_1) \subseteq \mathcal{O}(L_2)$ the load of label L_1 must be smaller than or equal to that of L_2.

In the labeling algorithm of Sol (1994), labels contain the cost c, the arrival time t, and the sets $S^+ \subseteq \{1,\ldots,n\}$ and $S^- \subseteq \{1,\ldots,n\}$. Here, S^+ is the set of requests that have been started and S^- is the set of requests that have been completed. With respect to the sets $\mathcal{V}$ and $\mathcal{O}$, one obtains $S^+ = \mathcal{V}$ and $S^- = \mathcal{V} \setminus \mathcal{O}$. In terms of S^+ and S^- the dominance criterion proposed in this paper is the following: label L_1 dominates label L_2 if

$$\eta(L_1) = \eta(L_2), \quad t(L_1) \leq t(L_2), \quad c(L_1) \leq c(L_2),$$

$$S^+(L_1) \subseteq S^+(L_2), \quad \left(S^+(L_1) \setminus S^-(L_1)\right) \subseteq \left(S^+(L_2) \setminus S^-(L_2)\right). \tag{32}$$

Sol (1994) used the following criterion:

$$\eta(L_1) = \eta(L_2), \quad t(L_1) \leq t(L_2), \quad c(L_1) \leq c(L_2),$$

$$S^+(L_1) = S^+(L_2), \quad S^-(L_1) = S^-(L_2). \tag{33}$$

Proposition 2. If label L_1 dominates label L_2 according to the criterion used by Sol (1994) then it also dominates L_2 using criterion (DOM1), but the converse is not true.

Proof. Obvious from conditions (32) and (33). □

It follows from Proposition 2 that the criterion DOM1 is less restrictive than the one used by Sol (1994).

Given a label L, let $U(L)$ be the set of *unreachable* requests from $\mathcal{P}(L)$. This set is defined as follows: $U(L) = \mathcal{V}(L) \cup \left\{i \in P : t(L) + t_{\eta(L),i} > b_i\right\}$. Augmenting the definition of unreachable requests by a condition on the end of the time window of the delivery of request i as follows $t(L) + t_{\eta(L),i} + t_{i,n+i} > b_{n+i}$ is not necessary due to the preprocessing steps described in Section 6.4. The preprocessing ensures that $b_i + t_{i,n+i} \leq b_{n+i}, \forall i \in P$.

By replacing $\mathcal{V}(L)$ with $U(L)$ in (31), one obtains the new dominance criterion (DOM1').

Proposition 3. DOM1' is a valid dominance criterion when d_{ij} satisfies the delivery triangle inequality.

Proof. In order to prove the validity of the new dominance criterion one has to consider the case where a label L_1 dominates a label L_2 according to the new criterion, but not according to (DOM1), i.e., when $U(L_1) \subseteq U(L_2)$ but $\mathcal{V}(L_1) \nsubseteq \mathcal{V}(L_2)$. Define $W = \mathcal{V}(L_1) \setminus \mathcal{V}(L_2)$. Any extension of $\mathcal{P}(L_1)$ cannot visit the requests in W. Hence, if an extension of $\mathcal{P}(L_2)$ could visit a request $i \in W$ then an extension of $\mathcal{P}(L_2)$ could be better than any extension of $\mathcal{P}(L_1)$. To see that no extension of $\mathcal{P}(L_2)$ can visit requests in W observe that $W \subseteq U(L_2)$ since $W \subseteq \mathcal{V}(L_1) \subseteq U(L_1) \subseteq U(L_2)$. As a result, any extension of $\mathcal{P}(L_2)$ that visits a node from W is violating a time window because of the definition of $U(L)$ and the assumption that t_{ij} satisfies the triangle inequality. □

Proposition 4. DOM1' is less restrictive than DOM1

Proof. It is easy to find examples that shows that $L_1 \preceq_{dom} L_2$ according to DOM1' does not imply $L_1 \preceq_{dom} L_2$ according to DOM1. On the other hand if $L_1 \preceq_{dom} L_2$ according to DOM1 then $L_1 \preceq_{dom} L_2$ according to DOM1' as well. To see this, note that $L_1 \preceq_{dom} L_2$ according to DOM1 implies, by definition, that $\eta(L_1) = \eta(L_2)$ and $t(L_1) \leq t(L_2)$. This implies that $\left\{i \in P : t(L_1) + t_{\eta(L_1),i} > b_i\right\} \subseteq \left\{i \in P : t(L_2) + t_{\eta(L_2),i} > b_i\right\}$. Since $L_1 \preceq_{dom} L_2$ according to DOM1 implies that $\mathcal{V}(L_1) \subseteq \mathcal{V}(L_2)$ we obtain that $U(L_1) = \mathcal{V}(L_1) \cup \left\{i \in P : t(L_1) + t_{\eta(L_1),i} > b_i\right\} \subseteq \mathcal{V}(L_2) \cup \left\{i \in P : t(L_2) + t_{\eta(L_2),i} > b_i\right\} = U(L_2)$. □

The idea of considering $U(L)$ instead of $\mathcal{V}(L)$ was proposed by Feillet et al. (2004) in the context of the pricing problem for the VRPTW. In the computational experiments reported in Section 7 of this paper, DOM1' was used.

The dominance criteria (DOM1) and (DOM1') are strong, but they give rise to strict requirements on the cost structure of the underlying network. The definition of d_{ij} from equation (19) cannot be used together with (DOM1) and (DOM1'). Indeed, one cannot ensure that the removal of a delivery node from a subpath will not increase the cost, and this property was used in Propositions 1 and 3. In Section 3.5 we show that this is not a major problem as any cost matrix can be transformed into one that satisfies the delivery triangle inequality while maintaining the cost of every feasible SP1 path. Another limitation of these dominance criteria is that the removal of arcs from the network must be performed very carefully. An arc (i, j) cannot be removed if the subpath (i, k, j) is valid for some delivery node k. In this case one cannot argue that removing deliveries from a path will yield a path with lower cost since removing the deliveries will result in an invalid route. Arc elimination is useful within a branch-and-bound scheme that branches on the arcs in the original formulation (1)-(12).

3.2.3 Label elimination

Dumas et al. (1991) have proposed rules for eliminating labels that cannot be extended to node $2n + 1$. The key observation is that given a label L one can examine the deliveries of the open requests in $\mathcal{O}(L)$. If it is impossible to create a path from $\eta(L)$ to node $2n + 1$ that goes through the deliveries of $\mathcal{O}(L)$ and that satisfies all time windows, then label L can be discarded because of the triangle inequality on t_{ij}. Determining whether such a path exists can be done by solving a traveling salesman problem with time windows, which is $\mathcal{NP}$-hard. Consequently, Dumas et al. (1991) have proposed to consider only subsets of $\mathcal{O}(L)$ of cardinality one and two. We use the same approach here. In addition, we also test two subsets containing three deliveries. In the first subset, the first delivery, i_1, is the one farthest from $\eta(L)$, the next delivery, i_2, is the one farthest from $\eta(L)$ and i_1, and the last delivery is the one farthest from $\eta(L)$, i_1 and i_2. In the second subset we chose the deliveries that have the earliest deadline b_i.

3.3 SPPTWCPD - SP2

We now consider the *Shortest Path Problem with Time Windows, Capacity, and Pickup and Delivery* (SPPTWCPD), denoted SP2, which relaxes SP1 by not requiring paths to be elementary. In this problem we do, however, impose two conditions which help prevent

cycles: *i)* after performing a pickup, the same pickup cannot be performed again before the corresponding delivery has been performed, and *ii)* a delivery cannot be performed before the corresponding pickup has been performed. These conditions ensure that any cycle in a path will contain at least four nodes. The shortest cycle is of the form $i \rightarrow n+i \rightarrow j \rightarrow i$. One cannot go from $n+i$ to i as the corresponding arc does not exist in our graph. If time windows are tight, such cycles are unlikely to arise and the SPPTWCPD should yield good lower bounds. This shortest path problem was used as a pricing problem by Dumas et al. (1991).

It is interesting to note that for the VRPTW the two commonly used pricing problems are the elementary shortest path problem with time windows and capacity constraints (ESPPTWC) and the shortest path problem with time windows and capacity constraints (SPPTWC) that relaxes ESPPTWC by allowing paths with cycles. Pseudo-polynomial algorithms for the SPPTWC are known (see for example Desrochers et al. (1992)) while Dror (1994) proved that the ESPPTWC is strongly $\mathcal{NP}$-hard which implies that no pseudo polynomial algorithm exists for the problem unless $\mathcal{P} = \mathcal{NP}$. For the PDPTW it was shown by Ropke and Pisinger (2007) that both the ESPPTWCPD and the SPPTWCPD relaxation are strongly $\mathcal{NP}$-hard.

3.3.1 Label management

In the labels for the SPPTWCPD we store η, t, l, c and $\mathcal{O}$ together with a pointer to the parent label, as explained in Section 3.2.1, but we do not store the set $\mathcal{V}$. Determining whether an extension of a label is feasible and creating new labels is done in a similar way as for SP1. We do not, however, maintain the set $\mathcal{V}$. Hence, equation (29) is not used and condition (22) is replaced with

$$0 < j \leq n \quad \wedge \quad j \notin \mathcal{O}(L). \tag{34}$$

Replacing condition (22) with (34) implies that non elementary paths can be generated. When the delivery of request i has been performed, i is removed from $\mathcal{O}$ according to equation (30) and the path may again visit the pickup node of request i.

3.3.2 Dominance criterion

The dominance criterion employed, denoted (DOM2), is the following: a label L_1 dominates a label L_2 if

$$\eta(L_1) = \eta(L_2), \quad t(L_1) \leq t(L_2), \quad c(L_1) \leq c(L_2), \quad \mathcal{O}(L_1) \subseteq \mathcal{O}(L_2).$$

Dumas et al. (1991) showed that this dominance criterion is valid. The concerns about distance matrix and missing arcs that were discussed in Section 3.2.2 apply to DOM2 as well. Finally, the label elimination rules described in Section 3.2.3 can also be used for the SPPTWCPD. It is actually in this context that they were first introduced by Dumas et al.

3.4 Possible improvements

Irnich and Villeneuve (2006) have proposed a labeling algorithm that solves non-elementary shortest path problems while ensuring that cycles of length k or smaller do not occur. Their

approach could be used to strengthen the lower bound of the LPM when using SP2 as a pricing problem since SP2 allows cycles containing more than two arcs. However, the computational results presented in Section 7 show that the lower bound obtained with SP2 already are quite close to the lower bounds obtained with SP1, so it is not clear if the effort involved with forbidding longer cycles is worthwhile.

On a different note, Righini and Salani (2006) have proposed a bi-directional approach to shortest path problems with resource constraints. Instead of starting the label extension only from the source node, they simultaneously extend labels both from the source and the sink nodes. The two searches eventually meet at a point where the paths from the source are merged with paths from the sink. This approach has shown great potential for reducing the running time of the shortest path algorithm. Although it is out of the scope of the current paper to apply this technique to the shortest path problems considered, it would be a promising area for future research.

3.5 Transforming the pricing problem cost matrix

In Sections 3.2 and 3.3 it was shown how effective dominance criteria could be devised for SP1 and SP2 when the cost matrix for the pricing problem satisfies the *delivery triangle inequality*. In Section 2.3 we saw that the pricing problem cost matrix does not satisfy the property when cuts have been added to the master problem. In this section we explain how it is possible to transform an arbitrary cost matrix into a cost matrix that satisfies the delivery triangle inequality while maintaining the optimal solutions of SP1 and SP2.

Lemma 1. For any vector $(\theta_1, \ldots, \theta_{|P|}) \in \mathbb{Q}^{|P|}$, let the cost matrix $(\tilde{d}_{ij})$ be defined by

$$\begin{array}{rclrcll} \tilde{d}_{ij} & = & d_{ij} - \theta_i, & \tilde{d}_{n+i,j} & = & d_{n+i,j} + \theta_i & \forall i \in P, \forall j \in N \\ \tilde{d}_{0j} & = & d_{0j} & & & & \forall j \in N. \end{array}$$

Using $(\tilde{d}_{ij})$ instead of (d_{ij}) does not change the cost of any feasible path in SP1 or SP2.

Proof. As any feasible path visiting node $i \in P$ also has to visit node $n + i \in D$ and vice versa the sum of the θ_i terms sums to zero on any feasible path. □

The next proposition shows how to select the modifiers θ_i such that $\tilde{d}_{ij}$ satisfies the delivery triangle inequality.

Proposition 5. If

$$\theta_j \geq d_{ik} - (d_{i,n+j} + d_{n+j,k}) \quad \forall j \in P, \forall i, k \in N$$

then $(\tilde{d}_{ij})$ defined in Lemma 1 satisfies

$$\tilde{d}_{ij} + \tilde{d}_{jk} \geq \tilde{d}_{ik} \quad \forall i, k \in N, \forall j \in D.$$

Proof. By distinguishing between the four cases $i = 0$, $i \in P$, $i \in D$, $i = 2n + 1$ (setting $\tilde{d}_{2n+1,j} = d_{2n+1,j}$ for all $j \in N$) and substituting the definition of $\tilde{d}_{ij}$ into the expression

$\tilde{d}_{ij} + \tilde{d}_{jk}$ we obtain the desired result. For example for $i \in P$ we obtain

$$\begin{aligned}
\tilde{d}_{ij} + \tilde{d}_{jk} &= d_{ij} - \theta_i + d_{jk} + \theta_{j-n} \\
&\geq d_{ij} - \theta_i + d_{jk} + \max_{i',k' \in N}\{d_{i'k'} - (d_{i'j} + d_{jk'})\} \\
&\geq d_{ij} - \theta_i + d_{jk} + (d_{ik} - (d_{ij} + d_{jk})) \\
&= d_{ik} - \theta_i \\
&= \tilde{d}_{ik}
\end{aligned}$$

for all $j \in D$ and $k \in N$. □

In practice one can use the modifiers $\theta_j = \max_{i,k \in N}\{d_{ik} - (d_{i,n+j} + d_{n+j,k})\}, \forall j \in P$ which can be calculated in $O(n^3)$ time. Note that the transformation ensures that $\tilde{d}_{ij}$ satisfies the delivery triangle inequality even if the original cost matrix (c_{ij}) does not.

4 Valid Inequalities

In this section we describe several families of valid inequalities that have been used in the branch-and-cut algorithms proposed by Cordeau (2006) and by Ropke et al. (2007). For two of these families, *rounded capacity inequalities* and *precedence inequalities*, we provide strengthened inequalities. We also show how the so-called *2-path cuts* for the VRPTW can be applied to the PDPTW. All of these inequalities are useful in strengthening the LP-relaxation of two-index and three-index formulations of the PDPTW, but as will be shown in Section 5 some of them are in fact implied by the LP relaxation of the set-partitioning formulations considered in this paper.

To describe these inequalities, it is convenient to introduce new notation. For any node subset $S \subseteq N$, let $\delta^+(S) = \{(i,j) \in A | i \in S, j \notin S\}$. We also use the notation $\delta^+(i)$ to designate the set $\delta^+(\{i\})$. Also let $\pi(S) = \{i \in P | n+i \in S\}$ and $\sigma(S) = \{n+i \in D | i \in S\}$ denote the sets of *predecessors* and *successors* of S. Finally, let $x_{ij} = \sum_{k \in K} x_{ij}^k$.

4.1 Infeasible path inequalities

Cordeau (2006) and Ropke et al. (2007) have discussed infeasible path inequalities and various strengthenings for the PDPTW. In this paper we use two types of infeasible path inequalities. Consider an infeasible path $R = (k_1, \ldots, k_\rho)$, then the inequality

$$\sum_{i=1}^{\rho-1} x_{k_i,k_{i+1}} \leq \rho - 2 \tag{35}$$

is valid.

If $k_1 = i$ and $k_\rho = n+i$ for some $i \in P$ and the path is infeasible because of time windows then Cordeau (2006) has observed that the inequality can be strengthened to

$$\sum_{i=1}^{\rho-1} x_{k_i,k_{i+1}} \leq \rho - 3. \tag{36}$$

4.2 Fork inequalities

Let $R = (k_1, \ldots, k_\rho)$ be a path in G and $S, T_1, \ldots, T_\rho \subset (P \cup D)$ be subsets such that $k_j \notin T_{j-1}$ for $j = 2, \ldots, \rho$. If for any integer $h \leq \rho$ and any node pair $i \in S, j \in T_h$, the path $(i, k_1, \ldots, k_h, j)$ is infeasible, then the following inequality is valid for the PDPTW:

$$\sum_{i \in S} x_{i,k_1} + \sum_{h=1}^{\rho-1} x_{k_h,k_{h+1}} + \sum_{h=1}^{\rho} \sum_{j \in T_h} x_{k_h,j} \leq \rho. \tag{37}$$

Similarly, let $R = (k_1, \ldots, k_\rho)$ be a path in G and $S_1, \ldots, S_\rho, T \subset (P \cup D)$ be subsets such that $k_j \notin S_{j+1}$ for $j = 1, \ldots, \rho - 1$. If for any integer $h \leq \rho$ and any node pair $i \in S_h, j \in T$, the path $(i, k_h, \ldots, k_\rho, j)$ is infeasible, then the following inequality is valid for the PDPTW:

$$\sum_{h=1}^{\rho} \sum_{i \in S_h} x_{i,k_h} + \sum_{h=1}^{\rho-1} x_{k_h,k_{h+1}} + \sum_{j \in T} x_{k_\rho,j} \leq \rho. \tag{38}$$

Inequalities (37) and (38) were introduced by Ropke et al. (2007) and are called *outfork* and *infork inequalities*, respectively. We refer to the paper by Ropke et al. (2007) for examples and figures.

4.3 Rounded capacity inequalities

Rounded capacity inequalities which are often used in the context of the VRP (see, e.g., Naddef and Rinaldi, 2002) can also be used for the PDPTW. For any node subset $S \subseteq P \cup D$, let $\kappa(S)$ be a lower bound on the number of times that vehicles must enter the set. The following inequality is then clearly valid:

$$x(\delta^+(S)) \geq \kappa(S). \tag{39}$$

Cordeau (2006) and Ropke et al. (2007) have proposed to use $\kappa(S) = \max\left\{1, \left\lceil \frac{|q(S)|}{Q} \right\rceil\right\}$. This lower bound can be improved to $\kappa(S) = \max\left\{1, \left\lceil \frac{q(\pi(S)\setminus S)}{Q} \right\rceil, \left\lceil \frac{-q(\sigma(S)\setminus S)}{Q} \right\rceil\right\}$. The bound is valid as $q(\pi(S)\setminus S)$ is a lower bound on the load of the vehicles entering set S and $-q(\sigma(S)\setminus S)$ is a lower bound on the load of the vehicles leaving S. This new lower bound is stronger than the previous one because

$$\begin{aligned} |q(S)| &= |q(S \cap D) + q(S \cap P)| \\ &= |q(\pi(S)) + q(\sigma(S))| \\ &= |q(\pi(S) \setminus S) + q(\sigma(S) \setminus S)| \\ &\leq \max\left\{q(\pi(S) \setminus S), -q(\sigma(S) \setminus S)\right\}. \end{aligned}$$

and examples can be constructed where the inequality is strict. The second equality holds because $q(S \cap D) = -q(\pi(S))$ and $q(S \cap P) = -q(\sigma(S))$, the third equality holds because $q(\{i, n+i\}) = 0$, and the inequality holds because $q(\pi(S) \setminus S) \geq 0$ and $q(\sigma(S) \setminus S) \leq 0$.

4.4 2-path inequalities

2-path inequalities are a variation of the rounded capacity inequalities and were proposed for the VRPTW by Kohl et al. (1999). When the set $S \subseteq P \cup D$ cannot be served by a single path, the following inequality is valid:

$$x(\delta^+(S)) \geq 2. \tag{40}$$

To determine whether the set S can be served by a single path one can go further than just considering capacities as in Section 4.3. Indeed, for the VRPTW Kohl et al. (1999) proposed to solve a TSPTW on the set of nodes: if the TSPTW is infeasible then at least two paths are needed to visit all nodes in S.

For the PDPTW a similar approach can be used. If it is impossible to find a tour serving all nodes in S while satisfying precedence, capacity and time window constraints then any feasible solution must use at least two arcs from the set $\delta^+(S)$. The idea can be taken further by observing that if a path serves all nodes in S by entering and leaving the set once, then the nodes $\pi(S) \setminus S$ must be served by this path before entering S and the nodes $\sigma(S) \setminus S$ must be served after leaving S. If such a path cannot be found then S defines a valid inequality (40) even though there exists a tour through S satisfying precedence, capacity and time window constraints.

4.5 Reachability inequalities

For any node $i \in N$, let $A_i^- \subset A$ be the minimum arc set such that any feasible path from the origin depot 0 to node i uses only arcs from A_i^-. Let also A_i^+ be the minimum arc set such that any feasible path from i to the destination depot $2n+1$ uses only arcs in A_i^+. Consider a node set T such that each node in T must be visited by a different vehicle. This set is said to be *conflicting*. For any conflicting node set T, define the *reaching arc set* $A_T^- = \cup_{i \in T} A_i^-$ and the *reachable arc set* $A_T^+ = \cup_{i \in T} A_i^+$. For any node set $S \subseteq P \cup D$ and any conflicting node set $T \subseteq S$, the following two valid inequalities were introduced by Lysgaard (2006) for the VRP with time windows:

$$x(\delta^-(S) \cap A_T^-) \geq |T| \tag{41}$$

$$x(\delta^+(S) \cap A_T^+) \geq |T|. \tag{42}$$

These inequalities are obviously also valid for the PDPTW. In this problem, however, nodes can be conflicting not only because of time windows but also because of the interactions with the precedence relationships and the capacity constraints.

4.6 Precedence inequalities

Let $S \subset N$ be a subset such that $i, 2n+1 \in S$ and $0, n+i \notin S$ for some $i \in P$. Then the following inequality is valid:

$$\sum_{(i,j) \in \delta^+(S)} x_{ij} \geq 1.$$

Precedence inequalities were introduced by Ruland and Rodin (1997) in the context of the TSP with pickup and delivery.

4.7 Strengthened precedence inequalities

Using ideas from the reachability inequalities, precedence inequalities can be strengthened as follows.

Proposition 6. Let A_i be the set of arcs that can be used in a feasible path from i to $n+i$. Furthermore let $S \subset N$ be a subset such that $i, 2n+1 \in S$ and $0, n+i \notin S$ for some $i \in P$. Then the following inequality is valid for the PDPTW:

$$\sum_{(i,j)\in(\delta^+(S)\cap A_i)} x_{ij} \geq 1. \tag{43}$$

Proof. Assume that inequality (43) is violated in a feasible integer solution. This implies that there exists a request $i \in P$ and a set $S \subset N$ such that $i, 2n+1 \in S$, $0, n+i \notin S$ and $\sum_{(i,j)\in(\delta^+(S)\cap A_i)} x_{ij} = 0$. Let p be the path in the solution that visits node i. To be part of a feasible solution, path p must visit node $n+i$ after visiting node i. Let p' be the subpath of p that starts in i and ends in $n+i$. It is clear that p' must use at least one arc from $\delta^+(S)$ because $i \in S$ and $n+i \notin S$. From the definition of A_i it is also clear that the arcs in p' all belong to the set A_i. Consequently, $\sum_{(i,j)\in(\delta^+(S)\cap A_i)} x_{ij} \geq 1$. □

5 Relationship Between Set Partitioning Formulations and Valid Inequalities

This section explores the relationship between the set partitioning formulations using SP1 and SP2 as pricing problems and the valid inequalities presented in Section 4. It turns out that several families of valid inequalities are implied by the set partitioning formulation with the SP1 subproblem. No such analysis was previously performed for the PDPTW, but some results were obtained by Letchford and Salazar González (2006) for set partitioning relaxations of the VRP and VRPTW. Such a study is not only interesting from a theoretical point of view, but it is also useful from a practical perspective as it implies that we do not need to consider separation procedures for certain classes of valid inequalities. Without the results presented in this section one would not know whether the inequalities implied by the set partitioning formulation were just inefficient for the instances considered while being useful for other instances.

Given a solution y^* to the LP relaxation of the set partitioning formulation given by (13), (14) and (16) one can obtain the corresponding two-index solution x^* given by $x^*_{ij} = \sum_{r\in\Omega} y^*_r \psi_{ijr}, \forall i, j \in N$, where ψ_{ijr} is a constant indicating the number of times arc (i,j) is used in route r. Let Ω^*_i be the set of columns serving request i in the current solution, i.e., $\Omega^*_i = \{r \in \Omega : y^*_r > 0, a_{ir} \geq 1\}$.

We first show that the fork inequalities are implied by SP1.

Lemma 2. Let p be a path generated by SP1 that visits $q \leq \rho$ nodes from the path $R = (k_1, \ldots, k_\rho)$. Path p can use at most q of the arcs in any valid outfork inequality defined on path R.

The lemma is illustrated in Figure 1. In this example the outfork inequality is defined for the path $R = (k_1, k_2, k_3, k_4)$ and the sets $S, T_1, \ldots, T_4$. The path $p = (v_1, k_2, k_3, v_2, v_3)$ (where $v_2 \in T_3$) visits two nodes from R and it uses the arcs (k_2, k_3) and (k_3, v_2) from the outfork inequality.

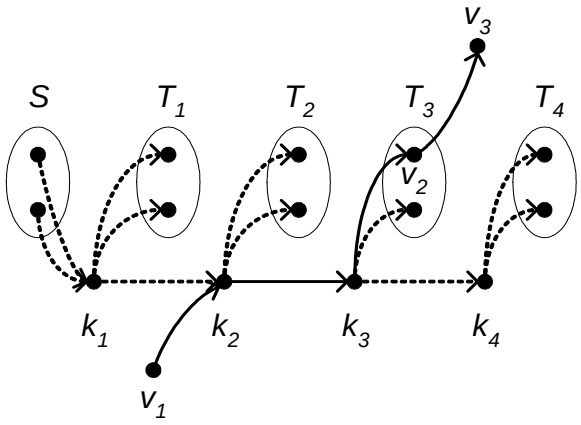

Figure 1: Outfork inequality for $R = (k_1, k_2, k_3, k_4)$ and the path $p = (v_1, k_2, k_3, v_2, v_3)$. Arcs in the outfork inequality are dashed while arcs in path p are solid. The arcs (k_2, k_3) and (k_3, v_2) are used in both the outfork inequality and in the path p

Proof. We break the path p into subpaths $p_1, \ldots, p_j$ by traversing p from its start and creating a new subpath p_w once a node from R is visited. We then continue to add nodes to p_w as long as p visits nodes from R. When p visits a node outside R, subpath p_w is ended and we start a new subpath p_{w+1} the next time p visits a node from R. Let $|p_w|$ be the number of nodes in subpath p_w ($|p_w| = 1$ is possible). To illustrate the concept of subpaths, consider Figure 2. In this example the paths p and R induce the subpaths $p_1 = (k_3, k_4, k_6)$ and $p_2 = (k_1, k_2)$.

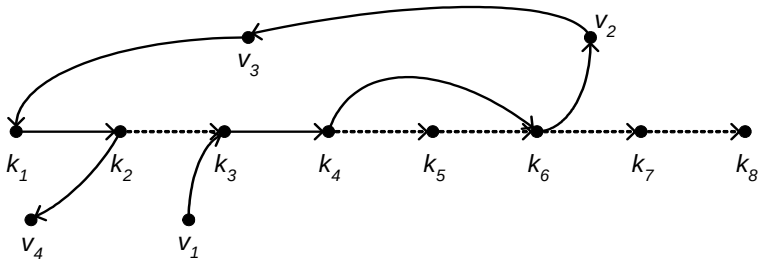

Figure 2: The paths $p = (v_1, k_3, k_4, k_6, v_2, v_3, k_1, k_2, v_4)$ (solid arcs) and $R = (k_1, \ldots, k_8)$ (dashed arcs).

It is clear that only the arcs used in the subpaths $p_1, \ldots, p_j$ along with the arcs that path p uses before entering or after leaving path $p_w, w \in \{1, \ldots, j\}$ can be present in the outfork inequality. Consider a subpath p_w that starts in node $k_i, i \neq 1$. The arc in p that is used

to enter path p_w cannot be part of the outfork inequality (because the arcs entering k_i in the outfork inequality originate from a node in R). Hence, p_w along with the arcs used to enter and leave p_w can contribute at most $|p_w|$ arcs from the outfork inequality as the path contains $|p_w|-1$ arcs and the arc used to leave the last node in p_w can be part of the outfork inequality. If a subpath p_w starts in node k_1 then the arc in p that is used to enter path p_w can be part of the outfork inequality. In that event we have to consider two cases: if $p_w = \left(k_1, \ldots, k_{|p_w|}\right)$ then the arc used to leave p_w cannot be part of the outfork inequality as this would make p_w infeasible. If p_w skips some of the first $|p_w|$ nodes from R then the arc used to leave p_w can be part of the outfork inequality but the first arc in p_w used to skip a node in R cannot be part of the outfork inequality as p is a feasible path. Consider for example $p_w = (k_1, k_2, k_5, k_6)$. This subpath skips node k_3 and k_4. If the arc used to enter p_w originates in S then (k_2, k_5) cannot be part of the outfork inequality as the original path p is feasible.

We see that p_w along with the arcs used to enter and leave p_w contributes at most $|p_w|$ arcs, for all $w \in \{1, \ldots, j\}$. Finally, we may conclude that a path that visits q nodes from path R can use at most q arcs from the outfork inequality defined on R. □

Proposition 7. If a vector y^* satisfies the inequalities (14) and (16) where Ω is defined by SP1 then the implied two-index solution x^* satisfies the outfork inequalities (37) defined in Section 4.2.

Proof. Consider a vector y^* that satisfies the conditions in the proposition and assume it violates an outfork inequality defined by a path $R = (k_1, \ldots, k_\rho)$ and sets $S, T_1, \ldots, T_\rho$. Let Φ_k^* be the set of paths from the solution y^* that visit exactly k nodes from the path R. From equation (14) we get that

$$\sum_{r \in \Phi_1^*} y_r^* + \sum_{r \in \Phi_2^*} 2y_r^* + \ldots + \sum_{r \in \Phi_\rho^*} \rho y_r^* = \rho. \tag{44}$$

From Lemma 2 we know that a path $p \in \Phi_k^*$ can use at most k arcs from the outfork inequality. Thus a path $p_k \in \Phi_k^*$ contributes at most $k y_{p_k}^*$ to the left-hand-side of equation (37). The total contribution from all paths visiting nodes in R is at most

$$\sum_{r \in \Phi_1^*} y_r^* + \sum_{r \in \Phi_2^*} 2y_r^* + \ldots + \sum_{r \in \Phi_\rho^*} \rho y_r^*,$$

which is equal to ρ according to (44). Consequently, the fork inequality cannot be violated. □

The set partitioning formulation using SP1 also implies all infork inequalities (38). The proof is similar to that of Proposition 7 and Lemma 2.

It is clear that the infeasible path inequality (35) is dominated by the fork inequalities and it is therefore implied by the set partitioning formulation using SP1. This infeasible path inequality is, however, not implied by SP2. As an example consider the path $(0, 1, n+1, 2, n+2, 1, n+1, 2, n+2, 2n+1)$. This is a feasible SP2 path and it can be used with value 0.5 in a feasible solution. In such a solution the infeasible path inequality $x_{n+1,2} + x_{2,n+2} + x_{n+2,1} \leq 2$

is violated as the left hand side is equal to 2.5. This also shows that the fork inequality is not implied by SP2.

The strengthened infeasible path inequality (36) is not implied by either formulation (small counter-examples can be constructed easily) and neither are the rounded capacity inequalities or the 2-path inequalities. The SP1 formulation does imply the reachability inequality as shown by the Proposition 8 below. We use the notation

$$\tau(i) = \begin{cases} i & : \ i \in P \\ i - n & : \ i \in D \end{cases}$$

to denote the request corresponding to a node $i \in P \cup D$.

Proposition 8. If a vector y^* satisfies the inequalities (14) and (16) where Ω is defined by SP1 then the implied two-index solution x^* satisfies the reachability inequalities defined in Section 4.5.

Proof. Assume that x^* violates the reachability inequality (41). This implies that there exists a set of conflicting nodes T and a set $S \subseteq P \cup D, T \subseteq S$ such that $\sum_{(j,k)\in(\delta^-(S)\cap A_T^-)} x^*_{jk} < |T|$. Consider a node $i \in T$ and a path p from $\Omega^*_{\tau(i)}$. The subpath p' of p that starts at node 0 and ends at node i uses only arcs from $A_i^- \subseteq A_T^-$ and it crosses $\delta^-(S)$ at least once as $0 \notin S$ and $i \in S$. Consequently every path corresponding to a column $r \in \Omega^*_{\tau(i)}$ contributes at least y^*_r to $\sum_{(j,k)\in(\delta^-(S)\cap A_T^-)} x^*_{jk}$. Because the paths are generated by SP1 we have that $\sum_{r\in\Omega^*_{\tau(i)}} y^*_r = 1$. This implies that that the paths serving i in solution y^* contribute at least 1 to $\sum_{(j,k)\in(\delta^-(S)\cap A_T^-)} x^*_{jk}$. As no path can serve two or more nodes in T we have $\Omega^*_i \cap \Omega^*_q = \emptyset$ for $i, q \in T, i \neq q$ and therefore $\sum_{(j,k)\in(\delta^-(S)\cap A_T^-)} x^*_{jk} \geq |T|$ which is a contradiction. The proof for inequality (42) is similar. □

A solution to the LPM using SP2 can violate reachability inequalities. The example used when discussing the relationship between SP2 and infeasible path inequalities also shows that reachability inequalities can be violated.

Proposition 9. If a vector y^* satisfies the inequalities (14) and (16) where Ω is defined by SP1 or SP2 then the implied two-index solution x^* satisfies the precedence inequalities defined in Section 4.6.

Proof. Assume that x^* violates a precedence inequality. This implies that there exist a request $i \in P$ and a set $S \subset N$ such that $i, 2n+1 \in S$, $0, n+i \notin S$ and $\sum_{(j,k)\in\delta^+(S)} x^*_{jk} < 1$. Every path in Ω^*_i uses at least one arc from $\delta^+(S)$ for every visit to node i as the path has to visit node $n+i$ before reaching node $2n+1$ or visiting node i again in case of SP2. Therefore, every column $r \in \Omega^*_i$ contributes at least $a_{ir}y^*_r$ to $\sum_{(j,k)\in\delta^+(S)} x^*_{jk}$. Because of equation (14) we have that $\sum_{r\in\Omega^*_i} a_{ir}y^*_r = 1$ and consequently, $\sum_{(j,k)\in\delta^+(S)} x^*_{jk} \geq 1$ which is a contradiction. □

Proposition 10. If a vector y^* satisfies the inequalities (14) and (16) where Ω is defined by SP1 or SP2 then the implied two-index solution x^* satisfies the strengthened precedence inequalities defined in Section 4.7.

Proof. The proof follows that of proposition 9. One should note that an SP1 or SP2 path that visits nodes $i \in P$ and $n+i$ only uses arcs from the set A_i between nodes i and $n+i$ and therefore each column $r \in \Omega_i^*$ contributes at least $a_{ir}y_r^*$ to $\sum_{(j,k)\in(\delta^+(S)\cap A_i)} x_{jk}^*$. □

6 Branch-and-Cut-and-Price Algorithm

In this section, we first describe heuristics for solving the pricing problem. This is followed by separation procedures for the identification of violated valid inequalities in Section 6.2 and by the branching strategy used to explore the enumeration tree in Section 6.3. Column- and cut-pool management is discussed in Section 6.4.

6.1 Pricing problem heuristics

It is well known that the running time of branch-and-price algorithms can be improved by using heuristic algorithms for the pricing problem. As long as the heuristic algorithms are able to find columns with negative reduced cost one can add those columns to the LPM and solve the problem again. Ideally it should be necessary to call the exact pricing algorithm only once for each node in the branch-and-bound tree to verify that no reduced cost column exists. In fact, this is not even necessary if the relaxation value associated with a node is lower than the current upper bound. In this case, the lower bound will not be used to fathom the node and it is not necessary to find the optimal relaxation value for this node. To use this strategy, however, one needs good pricing heuristics to avoid branching on a sub-optimal LP solution that is significantly different from the optimal one.

Using heuristic methods generally involves setting one or more parameters. Choosing the values for such parameters is often a trade-off between the solution quality and the running time of the heuristic. In this section we only report the value we chose for each parameter. To save space, we do not report on the experiments conducted in order to select these values. As a general rule, we have set the parameters so as to minimize the running time of the overall branch-and-cut-and-price algorithm.

We first present in Section 6.1.1 two heuristics obtained by truncating the label setting algorithms presented in Section 3. In Sections 6.1.2 and 6.1.3 we then introduce heuristics that follow the classic construction and improvement principle.

6.1.1 Label heuristics

It has often been proposed to turn exact labeling algorithms into heuristics by limiting the number of labels created in different ways. For example, Dumas et al. (1991) have proposed to reduce the network before running the pricing algorithm. They have created networks with between 30% and 50% of the "best" arcs. Irnich and Villeneuve (2006) have used a variant of this idea by creating reduced networks G_l where each node is connected to its l nearest neighbors.

In this paper we construct a graph G_l in which each node $i \in \{1,\ldots,2n\}$ is adjacent to at most l outgoing arcs reaching a pickup node and l outgoing arcs reaching a delivery node. We chose the arcs that are cheapest with respect to d_{ij}. After construction, all feasible arcs

of the form $(i, n+i), (0, i), (n+i, 2n+1)$ for $i \in P$ which are not already in G_l are added to G_l. Our implementation uses two reduced networks: G_5 and G_{10}. If the search using network G_5 does not find any path of negative reduced cost, then it switches to network G_{10}. The corresponding heuristic is denoted by $H1$.

Dumitrescu (2002) has proposed to limit the number of unprocessed labels at any time. In our context, this corresponds to putting a limit on U ($|U| \leq \mu$) in Algorithm 1. Only the μ best (with respect to reduced cost) labels are kept, the worst labels being discarded. This heuristic is used in a three-phase fashion. First, a limit $\mu = 500$ is used. If the heuristic does not find any negative cost path, then $\mu = 1000$, and finally $\mu = 2000$ is tried. This heuristic is denoted by $H2$.

For both of the heuristics and the exact algorithms based on the labeling algorithm we generate more than one negative reduced cost column if possible, but stop the algorithm if 100 negative reduced cost columns have been generated. All negative reduced cost columns are added to the LPM.

6.1.2 Construction heuristics

Sol (1994) has proposed to use a cheapest insertion heuristic to solve the ESPPTWCPD. A similar approach is implemented in this paper. Starting from a route containing only request i, we add the request that increases the reduced cost of the path the least. During insertions we keep track of the best route observed. The process is repeated with every request as a starting point. This algorithm is denoted by $H3$.

A straightforward way to improve this heuristic is by randomizing it. This can be done by performing insertions that are not the most promising: the possible insertions are ranked by insertion cost and a request is chosen by a random process that tends to select insertions with low cost. When using the randomized insertion it is worthwhile to try to construct a route starting several times from the same initial route containing request i. This algorithm is denoted by $H4$.

6.1.3 LNS heuristic

It is well known that improvement or steepest descent heuristics often produce high quality solutions in little time. This idea has been used by Sol (1994) and Savelsbergh and Sol (1998) to develop improvement heuristics for the pricing problem. As an initial solution to the pricing problem, routes corresponding to columns with a null reduced cost are used. The authors note that all columns used in the basis satisfy this condition. At each iteration in the improvement heuristic, one node is removed and another one is inserted in the path, thus leaving the length of the path unchanged.

In this section we describe a different improvement heuristic which is based on the Large Neighborhood Search (LNS) introduced by Shaw (1998). Ropke and Pisinger (2006) have shown that the LNS can be easily implemented by using simple construction heuristics, an idea that will be used here. The LNS algorithm attempts to improve an initial path by alternating between removing requests from the path and inserting requests into the path. The requests to remove are chosen randomly and requests are inserted using the randomized insertion algorithm outlined in Section 6.1.2.

The pseudo-code for the LNS is shown in Algorithm 2. The algorithm takes a path p and an integer σ as input. The parameter σ determines how many non-improving removal/insertion iterations should be performed before stopping the algorithm.

Algorithm 2 LNS pseudo code.

```
1  Input:  Path p, integer σ
2  for i = 1, ..., σ
3    f = ∞;
4    while (c(p) < f)
5      f = c(p);
6      p' = removeNodes(p);
7      p' = randomizedInsert(p');
8      if c(p') < f
9        p = p';
10 return p;
```

In line 3 we set f, the cost of the currently best solution. Line 4 makes the algorithm continue as long as an improvement is found. In line 6 nodes are removed from the path. The function `removeNodes`(p) returns a path where up to 50% of the nodes from p have been removed. In line 7 nodes are inserted into the path again by a randomized procedure that favors good insertions. In lines 8 and 9 the current solution is updated if an improvement was found. The function `randomizedInsert` is dependent on the shortest path problem that is solved (e.g., whether or not non-elementary paths are allowed).

The improvement heuristic is used in several contexts. In heuristic H5, LNS is used to improve the paths that are selected ($y_r > 0$) in the current LP solution and σ is set to 20. In heuristic H6, LNS is used to improve the paths generated by the randomized insertion heuristic described in Section 6.1.2. The LNS heuristic is applied to paths with reduced cost greater than or equal to 0 to try to bring the reduced cost below 0. In H6, σ is set to 5. In heuristic H7 LNS with $\sigma = 20$ is applied to all paths in the current LP solution to LPM that has reduced cost zero.

6.1.4 Using the pricing heuristics

Experiments performed by Ropke (2005) indicated that it is worthwhile to execute the heuristics in sequence, until a negative reduced cost path is found. The sequence should roughly be ordered by expected computation time of the heuristic. The exact algorithm is only called if all heuristics in the sequence fail. We use the following sequence of heuristics (H3, H5, H4, H2, H6, H7, H1).

6.2 Separation heuristics

In our implementation we use the following families of inequalities: infeasible path inequalities (36), fork inequalities, reachability inequalities, rounded capacity inequalities and 2-path inequalities. The precedence and strengthened precedence inequalities are not used as they

were shown to be implied by the set partitioning formulation in Section 5. We refer the reader to Ropke et al. (2007) for a description of the separation procedures for the infeasible path, fork and reachability inequalities. Below we describe how the new form of the rounded capacity inequalities as well as the 2-path inequalities are separated.

6.2.1 Separating rounded capacity inequalities

Rounded capacity inequalities are separated by a constructive heuristic which is called several times with different parameter settings in each call. The heuristic starts with a set S containing only one node (each node is tried as start node several times). Nodes are iteratively added to the set, performing the addition that most increases the objective f described below. The construction is stopped either when a violated inequality is detected or if the current set S is far from violating inequality (39). The objective we seek to maximize with each insertion is the following:

$$\begin{aligned} f(S) \;=\; & \lambda_1 \left(\max\left\{q(\pi(S)\setminus S), -q(\sigma(S)\setminus S)\right\} - Qx(\delta^+(S))\right) + \\ & \lambda_2 Q \left(\max\left\{\left\lceil\frac{q(\pi(S)\setminus S)}{Q}\right\rceil, \left\lceil\frac{-q(\sigma(S)\setminus S)}{Q}\right\rceil\right\} - x(\delta^+(S))\right) + \\ & \lambda_3 \left(\min\left\{q(\pi(S)\setminus S), -q(\sigma(S)\setminus S)\right\} - Qx(\delta^+(S))\right), \end{aligned}$$

where λ_1, λ_2 and λ_3 are parameters that control the importance of the three expressions. The reasoning behind the three expressions is the following: when the second expression is larger than 0 then we have discovered a violated inequality; the first expression can be seen as a weakened form of the second that better reflects changes in the demand of the nodes in S; the last expression emphasizes the part of the maximum expression that is not active in the two other expressions (in the hope that further augmentations of S may make this part active). In each call the parameters λ_1, λ_2 and λ_3 are chosen randomly from a uniform distribution on the set $[1,5]\times[1,5]\times[0,1]$, thus giving a higher weight to the two first terms.

6.2.2 Separating 2-path inequalities

The separation heuristic for the 2-path inequality is a randomized greedy construction heuristic. Starting from a set S containing only one node the heuristic augments S by adding the node that minimizes $x(\delta^+(S))$. In order to randomize the heuristic a random number from the interval $[0, 0.3]$ is added to each x_{ij} when selecting the node to add. For every set S for which $x(\delta^+(S)) < 2$ we check whether a feasible solution can leave S exactly once. To this purpose, we determine whether there exists a feasible path that first visits all the nodes in $\pi(S)\setminus S$, then visits all nodes in S and finally visits all nodes in $\sigma(S)\setminus S$ while satisfying time window, capacity and precedence constraints. This is determined by a straightforward, exact label-setting algorithm. If such a path does not exists then a valid 2-path inequality has been found. The augmentation of S stops when $x(\delta^+(S)) > 3$. All nodes are considered as start nodes and the heuristic is applied several times for each start node. Every time the algorithm has checked if a set S induces a 2-path inequality, the result is stored in a hash-table. This hash-table is examined before checking a set S for feasibility. Thereby we ensure that the same calculation is performed only once.

6.3 Branching strategy

When the solution of the LPM is fractional and no violated inequality can be identified, one has to resort to branching. Branching in a column generation algorithm should be done with care as the branching strategy should preferably be compatible with the algorithm used for solving the pricing problem, i.e., the same type of pricing problem should be solved in the children nodes as in the parent node. This implies that branching decisions should be easily transferred to the subproblem and should not change its structure. In our algorithm we use two branching rules that add a single cut on the x_{ij} variables to the master problem. They are thus compatible with the pricing problems when applying the transformation of the pricing problem described in Section 3.5.

The first branching rule branches on the outflow of a set of nodes as proposed for the VRP by Naddef and Rinaldi (2002). A set of nodes S is first selected such that $x(\delta^+(S))$ is as far as possible from the nearest integer. Two branches are then created: $x(\delta^+(S)) \leq \lfloor x(\delta^+(S)) \rfloor$ and $x(\delta^+(S)) \geq \lceil x(\delta^+(S)) \rceil$. In our implementation, a good candidate for the set S is found using a simple greedy heuristic, which most often finds a set containing only two nodes.

The second branching rule calculates $x(\delta^+(0))$. If $x(\delta^+(0))$ is fractional then the two branches $x(\delta^+(0)) \leq \lfloor x(\delta^+(0)) \rfloor$ and $x(\delta^+(0)) \geq \lceil x(\delta^+(0)) \rceil$ are created. This rule was proposed by Desrochers et al. (1992) and is often called *branching on the number of vehicles*.

In our implementation we first try to branch on the number of vehicles. If this is impossible because $x(\delta^+(0))$ is integer then we use the first branching rule instead. This choice was motivated by computational experiments documented in Ropke (2005).

In our branch-and-cut-and-price algorithm, the enumeration tree is explored in a depth-first fashion. We prefer depth-first compared to a best-bound strategy as it uses less memory and each node in the branch-and-bound tree is evaluated slightly faster as we have to perform less work in order to restore a valid basis and to populate the model with useful rows and columns when moving to a new node in the branch-and-bound tree. Furthermore, the depth-first strategy is the easiest to implement. The most significant drawback of the depth-first strategy is that it may visit more nodes than the best-bound strategy if the initial upper bound is poor. Our upper bounds are found using the heuristic described in Ropke and Pisinger (2006) and are usually quite tight (See Section 7 and Table 2).

6.4 Implementation issues

Every time the LPM has been solved we are faced with a choice of either trying to generate more variables (columns) or valid inequalities (rows). In our approach, column generation is performed as long as the heuristics are able to identify promising variables. When the heuristics fail, the cut generation routines take over unless the current LP value is above the upper bound or cut generation has been tried before without success at the current branch-and-bound node. If cut generation is tried and violated inequalities are identified, they are added to the model and the pricing heuristics are called again to identify more variables with negative reduced cost. If the cut generation is unsuccessful or if it is not performed (for the reasons mentioned above) then the exact pricing algorithm is called.

Several preprocessing rules for tightening time windows in the PDPTW have been described by Dumas et al. (1991) and Cordeau (2006). All these rules have been implemented

here. In addition, we apply the rules proposed by Desrochers et al. (1992) for tightening time windows in the VRPTW.

6.4.1 Cut pool management

Every time the LPM is solved the algorithm determines which of the valid inequalities previously generated are satisfied at equality by the current solution. If an inequality has not been binding for ten consecutive iterations, it is removed from the problem and inserted in a cut pool. Every time the LPM is solved, the cut pool is checked for violated inequalities. If a violated inequality is found in the cut pool, it is then added to the linear relaxation and the problem is solved again. Once an inequality has been identified by one of the separation procedures it is kept in memory, either explicitly in the model, or implicitly in the cut pool. In computational experiments, a significant performance boost was observed when using this approach compared to keeping all inequalities in the formulation.

6.4.2 Column pool management

It has been observed that keeping all columns in the LPM can be a computational burden when solving the LP relaxation. Several authors (e.g., Sol (1994), Savelsbergh and Sol (1998) and Lübbecke (2001)) have proposed to keep only a fraction of the generated columns in the LPM and maintain a pool of columns that are not needed in the current LP solution, but may be needed in the future. In order to decide whether a column should be kept in the pool or discarded completely its reduced cost is inspected.

In our implementation we also maintain a column pool. Columns are moved to the pool when the LPM becomes too large. When the LPM reaches 2000 columns we apply a method that moves to the pool columns that have not been part of the basis for a certain number of iterations and have a reduced cost larger than a certain threshold. The pool is always checked for columns with negative reduced cost before calling the heuristic or the exact pricing algorithms. Once a column has been generated it is thus always kept in memory, either in the LPM or in the column pool. As new inequalities are generated dynamically the columns that enter into the pool may differ from those that need to be retrieved from the pool at a later stage. Consequently we store paths instead of columns in the pool. It is easy to transform a path into the correct column, taking all currently active cuts into account.

Adding columns or rows to the LPM leaves the current solution either sub-optimal or infeasible. We do not try to exploit this by forcing the LP solver to use either a primal or dual simplex algorithm. Instead, for simplicity, we let the LP solver choose the appropriate algorithm automatically.

7 Computational Experiments

This section describes the computational experiments that we have performed to assess the performance of our branch-and-cut-and-price algorithm. The algorithm was implemented in C++ and run on an AMD Opteron 250 computer (2.4 GHz) running Linux. CPLEX 9.0 was used as LP solver and the COIN-OR Open Solver Interface

(OSI, `http://www.coin-or.org/index.html`) was used as an interface to the LP solver. In all experiments, a limit of two hours of CPU time was used unless otherwise indicated.

The algorithm was tested on two sets of instances: instances similar to those introduced by Ropke et al. (2007) (data set one) and the instances proposed by Li and Lim (2001) (data set two). The instances proposed by Li and Lim originate from the Solomon VRPTW instances (Solomon, 1987). For these instances we minimize the total traveled distance in our objective. The Li and Lim instances are divided into two series. Those in the first series have a short planning horizon while those in the second have a longer horizon allowing many requests to be served in the same route. We only report computational results for the first series as only a small subset of the second one could be solved. The pricing problems occurring in the second series are very difficult and the branch-and-cut-and-price algorithm often times out before even obtaining a lower bound at the root node. The integrality gap, on the other hand, was small for the instances that could be solved. See Ropke (2005) for some results on these instances.

The instances introduced by Ropke et al. were produced with a generator initially proposed by Savelsbergh and Sol (1998). As explained by Ropke et al. (2007), the generator was modified to obtain harder instances by reducing the ratio between the travel times and the length of the planning horizon. In addition, the new generator considers a single depot located at the middle of a square instead of a different depot for each vehicle. In all instances, the coordinates of each pickup and delivery location are chosen randomly according to a uniform distribution over the $[0, 50] \times [0, 50]$ square. The load q_i of request i is selected randomly from the interval $[5, Q]$, where Q is the vehicle capacity. A planning horizon of length $T = 600$ is considered and each time window has width W. The time windows for request i are constructed by first randomly selecting a_i in the interval $[0, T - t_{i,n+i}]$ and then setting $b_i = a_i + W$, $a_{n+i} = a_i + t_{i,n+i}$ and $b_{n+i} = a_{n+i} + W$. In all instances, the primary objective consists of minimizing the number of vehicles, and a fixed cost of 10^4 is thus imposed on each outgoing arc from the depot. The instance generator used by Ropke et al. (2007) contained a bug such that a_{n+i} was set to $b_i + t_{i,n+i}$ instead of $a_i + t_{i,n+i}$. This is unfortunate as it implies that cycling cannot occur with the SP2 algorithm on these instances because the pickup and delivery of each request have non-overlapping time windows. Consequently, we have generated new instances with a corrected version of the instance generator. The new instances appear to be more difficult compared to the instances considered by Ropke et al. (2007).

Four groups of instances were generated by considering different values of Q and W. The characteristics of these groups are summarized in Table 1. As in Ropke et al. (2007) we considered ten instances with $30 \leq n \leq 75$ for each group. The name of each instance (e.g., AA50) indicates the class to which it belongs and the number of requests it contains. We repeat the first letter in each instance to distinguish these instances from the ones studied by Ropke et al. (2007). The maximum travel time between two nodes in these instances is $\sqrt{50^2 + 50^2} \approx 70.7$ and the time windows are therefore relatively wide, especially for the CC and DD instances.

For all experiments we use the same numerical precision as Ropke et al. (2007), i.e., travel times and costs are represented using double precision floating point numbers. Upper bounds are found using the adaptive large neighborhood heuristic proposed by Ropke and Pisinger (2006). Table 2 shows the solutions found by the heuristic. The table is split into

Table 1: Characteristics of the new PDPTW instances

Class	Q	W
AA	15	60
BB	20	60
CC	15	120
DD	20	120

two major columns. The first major column shows results for data set one and the second major column shows results for the second data set. The first 30 instances in data set two contain 100 requests while the last 6 instances contain 500 requests. For each instance we provide three numbers. 1) A lower bound (LB) on the optimal solution. If the instance has been solved to optimality we report the optimal solution in bold, otherwise we report the best lower bound obtained at the root node by the SP1 or SP2 relaxation. If neither the SP1 nor the SP2 relaxation obtained a lower bound then this entry is left blank. We do not report the lower bound from the branch-and-cut procedure proposed by Ropke et al. (2007) because it usually is quite poor for these hard instances. 2) The solution value found by the heuristic (Heur). If the value is known to be optimal, it is shown in bold. This value, increased by 0.1, is used as an upper bound when testing the branch-and-cut-and-price algorithm in the following sections. We increase the value by 0.1 to ensure that the branch-and-cut-and-price algorithm finds a solution even when the heuristic value was optimal. 3) The time in seconds (Time) the heuristic spent finding the reported solution. The time spent by the heuristic is not included in the computation times reported in the following sections. Notice that the there often is a quite large integrality gap for the instances from the first data set that have not been solved to optimality (e.g. AA75). We believe that the main reason for this gap is a poor lower bound rather than a weak upper bound.

Our computational experiments focus on three aspects. First, we wanted to measure the impact of the valid inequalities described in Section 4. Second, we wished to investigate the impact of the two subproblems, SP1 and SP2 described in Section 3, on the lower bound and overall solution time. Third, we wanted to compare the performance of our branch-and-cut-and-price algorithm to the branch-and-cut algorithm of Ropke et al. (2007).

7.1 Impact of valid inequalities

The purpose of this section is to investigate the impact of the valid inequalities on the two set partitioning relaxations.

Table 3 reports results obtained by running the algorithm on the first data set. The table shows results for the SP1 and SP2 relaxations. The column *No cuts* indicates the value of the lower bound as a percentage of the upper bound. It is computed as $100\underline{z}'/\bar{z}$ where $\underline{z}'$ is the lower bound without adding any cuts and $\bar{z}$ is the upper bound (either the optimal solution or the best known solution if the optimal solution is unknown). The rest of the columns report the amount of gap closed using the different families of inequalities: *IPC* – Infeasible path constraints (36), *CC* – Rounded capacity constraints, *2PC* – 2-path

constraints, *FC* – Fork constraints, *RC* – Reachability constraints, *Full* – all constraints. These values are computed as follows: $100(\underline{z} - \underline{z}')/(\bar{z} - \underline{z}')$ where $\underline{z}$ is the lower bound found using the corresponding cuts. Some entries are left blank for instance AA45 because the relaxation without cuts solved the instance to optimality. Note that in some cases the entry in the column *Full* is worse than the value in one of the preceding columns (e.g., AA65, SP1). This would not happen if all inequalities were separated by exact algorithms, but can happen when inequalities are separated by heuristics.

The table shows a difference between the SP1 and SP2 relaxations although the difference is rather small. One also notes that the integrality gap at the root node is quite large for many instances (e.g., BB30). The large gap occurs because of the fixed cost on each vehicle and because the LP relaxation is unable to estimate the number of vehicles correctly, i.e., $x(\delta^+(0))$ is lower in the LP relaxation than in a feasible integer solution. One sees that the capacity and 2-path constraints are able to improve the lower bound somewhat, but none of the inequalities are able to improve the lower bound significantly. Future research should aim at finding valid inequalities that increase $x(\delta^+(0))$ in fractional solutions when possible.

Table 4 shows results obtained by applying valid inequalities to the Li and Lim instances with 100 requests. We only report results for the instances for which the lower bound without valid inequalities could be computed within 2 hours and which had a non-zero integrality gap for both relaxations when using a pure column generation approach. When adding valid inequalities, the computing times sometimes exceeded 2 hours. The blank entries for instance LR1_2_6 indicate that the SP1 lower bound was optimal without adding cuts.

For this set of instances we also observe a small difference between SP1 and SP2. We see that the valid inequalities are much more useful for this data set and the 2-path cuts are especially beneficial. It is interesting to see that the capacity cut is ineffective on this set of instances, the reason being that the capacity constraints are quite loose in these instances as opposed to those in the first data set.

7.2 Branch-and-cut-and-price experiments

This section compares the two set partitioning relaxations to each other as well as to the branch-and-cut algorithm proposed by Ropke et al. (2007). We also test the limits of the algorithms in terms of the instance size that can be solved to optimality. All valid inequalities presented in Section 4 and which are not implied by the relaxations are added dynamically to the model.

The results for the first data set are shown in Table 5. The first two columns show the instance name and the best known upper bound (the optimal solution value in the case of instances solved to optimality and the heuristic value from Table 2 otherwise). The remaining columns indicate $\underline{z}$ – the root node lower bound after adding cuts, *Time* – the total amount of CPU time used (in seconds), *Nodes* – the number of nodes in the branch-and-bound tree, and *Cuts* – the number of cuts added. If the problem was not solved within the time limit the entry in the *Time* column is left blank. If the lower bound could not be computed within the time limit the entry in the $\underline{z}$ column is left blank as well. The tables show results for the branch-and-cut algorithm (B&C) and the branch-and-cut-and-price algorithms using SP1 and SP2 as pricing problems. The row *Solved* indicates how many instances were solved to optimality within the time limit.

Both SP1 and SP2 produce good results for these instances and clearly outperform the branch-and-cut algorithm. The set partitioning formulation using SP1 and SP2 is much better at approximating the number of vehicles in the LP relaxation. The difference in performance between SP1 and SP2 is relatively small but we see that SP1 solves to more instances than SP2 and in general uses fewer branch and bound nodes.

Even though the SP1 and SP2 relaxations do very well, they fail to solve the largest instances and the instances with wide time windows prove to be difficult. Globally, 10 out of the 40 instances are unsolved. It is our experience that the fixed costs on each vehicle made the instances more difficult to solve. The branch-and-cut code from Ropke et al. (2007) is clearly outperformed as it only manages to solve 12 instances within the time limit.

Table 6 shows results for Li and Lim (2001) instances in the first series and with approximately 100 requests. Li and Lim (2001) also proposed instances with 50 requests, but the first series of these instances do not pose any difficulty for the branch-and-cut-and-price algorithm as all instances can be solved in less than 10 minutes (see Ropke (2005)). The instances with 100 requests are naturally harder to solve and 13 out of the 30 instances could not be solved to optimality within the time limit. One notices that no lower bound was found for several of these instances. This indicates that the pricing problem was too difficult. The SP1 relaxation manages to solve one more instance to optimality compared with the SP2 relaxation, but overall they behave similarly.

To test the algorithms on large instances we selected instances with approximately 500 requests from the Li and Lim dataset. We chose the six instances that have the tightest time windows. The results of this experiment are shown in Table 7. Both relaxations are able to solve three instances with 500 requests. To the best of our knowledge, these are the largest PDPTW instances solved to optimality in the literature.

7.3 Further experiments

This section provides some insights into the inner workings of the branch-and-cut-and-price algorithm that the two preceding sections omitted. For this purpose we select a small set of diverse instances that we use for further testing. This set contains CC45 and DD45 from the first data set and LR1_2_6, LR1_2_9 and LRC1_2_2 from the second data set. We also created two new instances XX45 and YY45. Those instances were created by the instance generator used for the first data set and both instances contain 45 requests. The XX45 instance uses parameters $Q = 15$ and $W = 60$ while the YY45 instance uses $Q = 15$ and $W = 120$. The demand q_i of every request in both instances is set to 1 such that the capacity constraints are very loose. These two instances were created to examine more loosely constrained instances similar to the instances from the first data set.

The first experiment aims at determining the effect of the valid inequalities in a branch-and-bound setting as opposed to the experiment in section 7.1 that tested the effect on the lower bound in the root node. The results of this experiment are shown in Table 8. The table contains three major columns. The first major column shows the lower bound at the root node, reported as $100\underline{z}/\bar{z}$, where $\underline{z}$ is the lower bound at the root node and $\bar{z}$ is the upper bound. The second one shows the time in seconds needed to solve the instance to optimality while the last one shows the number of branch-and-bound nodes explored. For each major column we list results for the SP1 and SP2 relaxations with cuts ($+cuts$) and

without cuts (*-cuts*). Blank entries correspond to instances that could not be solved within the two hour time limit. If an instance could not be solved to optimality we report the number of branch-and-bound nodes explored within the time limit, not the number of nodes needed to reach optimality. Each row corresponds to one of the seven instances while the last row averages each column (we exclude instances YY45 and LR1_2_9 from the averages in the time column).

The results show that even though the valid inequalities only improve the lower bound at the root node slightly, they have a significant, positive effect on the running time of the entire branch-and-cut-and-price algorithm and on the number of nodes explored as well. Most notable is the LR1_2_9 instance that cannot be solved within the time limit without the valid inequalities. The SP2 relaxation seems to be slightly more dependent on valid inequalities compared to the SP1 relaxation. This is natural as the initial integrality gap is largest for SP2.

The purpose of the second experiment is to shed light on how time is spent within the branch-and-cut-and-price algorithm. We solved the root node for each of the seven instances with both relaxations. The experiment is summarized in Table 9. The columns should be interpreted as follows: *PP*, pricing problem used; *Total time*, time spent solving the root node. *Time preproc.*, time spent on preprocessing, this includes the time spent tightening time windows (see Section 6.4) and calculating the sets A_i^+ and A_i^- needed for the reachability constraint (see Section 4.5); *Time LP*, time spent solving the linear programming relaxation; *Time PP heur.*, time spent solving the pricing problem by heuristic means; *Time PP exact*, time spent solving the pricing problem by the exact method; *Time sep.*, time spent separating valid inequalities; *#Cuts*, number of violated inequalities detected; *#Cols*, number of columns generated; *#PP heur*, number of calls to the pricing heuristic (this is equivalent to the number of iterations in the column generation method); *#PP exact*, number of calls to the exact pricing procedure; *#Labels*, the number of labels processed in the last call to the exact pricing procedure. We count a label as processed when it is being retrieved from the set of unprocessed label (line 4 in Algorithm 1). Every two lines in the table represent information about a particular instance. The first line is for the SP1 relaxation while the second line is for the SP2 relaxation. The last two lines provide averages.

All time measurements are in seconds. Notice that the sum of the partial time consumptions does not add up to the total time consumption reported. This is because time is spent in other parts of the algorithm (e.g. management of column and cut pools) that is not reported in the table. We see that the SP2 relaxation spends more time on preprocessing compared to SP1, the reason being that the reachability cuts that are turned of for SP1 as they are implied by this relaxation. The reachability cut requires the computation of the sets A_i^+ and A_i^- which can be time consuming, especially for larger instances like the last three. Running the separation procedure for the reachability constraints can also be time consuming as it involves many maximum flow calculations (see Lysgaard (2006)). This is especially visible for instance LRC1_2_2.

One sees that for these instances, solving the LP problems is an easy task that only accounts for a small fraction of the entire time spent, partly because of the low number of columns generated. The algorithm usually spends most of the time solving the pricing problem heuristically and exactly. Solving the pricing problem to optimality can be difficult, the best example is instance LR1_2_6. The table shows that only a few calls to the exact

pricing algorithm are necessary, which indicates that the pricing heuristics are working well. However, instance YY45 is an exception.

The table also shows that the SP2 relaxation usually needs more column generation iterations and that more columns usually are generated for this model. We believe that this is because the SP2 pricing problem is less constrained and the set of columns we are optimizing over is larger than for the SP1 relaxation. The comparison between SP1 and SP2 in terms of number of processed labels does not provide a clear picture. A priori one might have expected the SP2 pricing problem to be easier to solve than SP1, but as the results in Table 9 and in Section 7.2 show, this is not always the case.

The last experiment shows the effect of the pricing heuristics. In this experiment we compare the sequence of heuristics described in Section 6.1.4 to a single heuristic H1 (see Section 6.1.1). The single heuristic is comparable to pricing heuristics that often are used in the branch-and-price literature. In the experiment we solved the root node relaxation. The results of this experiment are summarized in Table 10. The table contains three major columns. The first reports the time for solving the root node, the next reports the number of calls to the pricing heuristic and the last reports the number of calls to the exact pricing procedure. The table contains four sub-columns for each major column. Here *SP1 std.* refers to pricing problem SP1 and the standard sequence of heuristics while *SP1 simple* refers to pricing problem SP1 solved with heuristic H1. *SP2 std.* and *SP2 simple* have similar interpretations for SP2. Each row in the table corresponds to an instance. The blank entry for LR1_2_6 and SP1 coupled with heuristic H1 indicates that the root node could not be solved within the 2 hour time limit. The number of calls to the pricing heuristic and to the exact pricing procedure for LR1_2_6 and SP1 coupled with heuristic H1 only shows the number of calls performed within the time limit. We would have seen higher numbers if we allowed the computation to run to the end. The last row averages each column but values for instance LR1_2_6 are not included in that computation.

The experiment shows that the advanced pricing heuristic is worthwhile, especially for SP1. One sees that the number of calls to the exact pricing procedure is greatly reduced with the advanced heuristic and this clearly pays off when the pricing problem is difficult as for instance LR1_2_6. If the test set had contained more instances with difficult pricing problems we would have seen an even larger difference between the performance of the simple and advanced heuristics. For some instances where the pricing problem is easy the simple heuristic performs best. This is true for CC45, DD45 and XX45. This indicates that we probably could do better on the first data set if we selected a sequence of heuristic that was tailored for these instances. We have avoided this to keep the results easier to understand. Another effect of using the simple pricing heuristic is that the number of column generation iterations increases. This causes the number of column generated to increase (2017 on average for *SP1 std.* versus 4777 on average for *SP1 simple*). The increased number of columns make the linear programming relaxations harder to solve, but for the instances considered, this is not a problem (0.7s on average for *SP1 std.* versus 2.1s on average for *SP1 simple*).

8 Conclusions

This paper has introduced a branch-and-cut-and-price algorithm for the PDPTW. We have proved that the most useful valid inequalities (fork and reachability inequalities) from a recently proposed branch-and-cut algorithm (Ropke et al. (2007)) are implied by the set partitioning relaxation SP1. Those inequalities are not implied by the SP2 relaxation, but despite this, the SP1 and SP2 relaxations provide similar lower bounds and are roughly equally hard to compute (for the instances considered in this paper). Even though both the SP1 and SP2 relaxations have been used in the literature before, this paper is the first to present a computational comparison between the two relaxations.

For the instances considered in this paper we recommend using the elementary shortest path problem as pricing problem. The pricing problem allows stronger lower bounds and it does not seem to be much harder to solve compared to its non-elementary counterpart. Furthermore, it seems easier to theoretically analyze a set partitioning relaxation based on elementary shortest paths.

In this paper we have introduced a number of significant improvements to existing algorithms for the ESPPTWCPD. We have also shown how adding valid inequalities from the 2-index formulation to the master problem interferes with the pricing algorithm. An approach to modify the cost matrix used in the pricing algorithm was proposed to ensure that the strongest dominance criteria could be used in the pricing algorithms while adding valid inequalities to the master problem.

The experiments concerning valid inequalities showed that the 2-path cut was the most successful of the valid inequalities tested. More research is needed in order to find new families of valid inequalities that close even more of the gap between the lower and upper bounds.

We may conclude that several large scale instances can be solved with the current approach but loosely constrained instances remain a tough challenge.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada under grant 227837-04. This support is gratefully acknowledged. We are also grateful to David Pisinger and Stefan Irnich for their valuable comments and suggestions.

References

E. Balas, M. Fischetti, and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68:241–265, 1995.

C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving integer programs. *Operations Research*, 46: 316–329, 1998.

J. Bramel and D. Simchi-Levi. Set-covering-based algorithms for the capacitated VRP. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs*

on Discrete Mathematics and Applications, chapter 9, pages 85–108. SIAM, Philadelphia, 2002.

J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54:573–586, 2006.

J.-F. Cordeau, G. Laporte, J.-Y. Potvin, and M.W.P. Savelsbergh. Transportation on demand. In C. Barnhart and G. Laporte, editors, *Transportation*, Handbooks in Operations Research and Management Science, pages 429–466. Elsevier, Amsterdam, 2007.

G. Desaulniers, J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis, and D. Villeneuve. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T.G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 57–93. Kluwer, Norwell, MA, 1998.

G. Desaulniers, J. Desrosiers, A. Erdmann, M.M. Solomon, and F. Soumis. VRP with pickup and delivery. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, chapter 9, pages 225–242. SIAM, Philadelphia, 2002.

M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6:301–325, 1986.

M. Dror. Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, 42(5):977–978, 1994.

Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

I. Dumitrescu. *Constrained Path and Cycle Problems*. PhD thesis, Department of Mathematics and Statistics, The University of Melbourne, May 2002.

D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, chapter 2, pages 33–65. Springer, 2005.

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(3):391–406, 2006.

N. Kohl, J. Desrosiers, O. B. G. Madsen, M.M. Solomon, and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101 – 116, 1999.

A.N. Letchford and J.J. Salazar González. Projection results for vehicle routing. *Mathematical Programming, Ser. B*, 105:251–274, 2006.

H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *The 13th IEEE International Conference on Tools with Artificial Intelligence, ICTAI-2001*, Dallas, USA, 2001.

Q. Lu and M. Dessouky. An exact algorithm for the multiple vehicle pickup and delivery problem. *Transportation Science*, 38:503–514, 2004.

M. Lübbecke. *Engine scheduling by column generation.* PhD thesis, Techniscen Universtät Braunschweig, 2001.

J. Lysgaard. Reachability cuts for the vehicle routing problem with time windows. *European Journal of Operational Research*, 175(1):210–223, 2006.

D. Naddef and G. Rinaldi. Branch-and-cut algorithms for the capacitated VRP. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, chapter 3, pages 53–84. SIAM, Philadelphia, 2002.

G. Righini and M. Salani. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006.

S. Ropke. *Heuristic and exact algorithms for vehicle routing problems.* PhD thesis, Department of Computer Science (DIKU), University of Copenhagen, 2005.

S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

S. Ropke and D. Pisinger. Complexity of shortest path problems arising in column generation for vehicle routing problems with pairing and precedence constraints. Working paper, 2007.

S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

K.S. Ruland and E.Y Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers and Mathematics with Applications*, 33(12):1–13, 1997.

M.W.P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29:17–29, 1995.

M.W.P. Savelsbergh and M. Sol. DRIVE: Dynamic routing of independent vehicles. *Operations Research*, 46:474–490, 1998.

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming)*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431, 1998.

M. Sigurd, D. Pisinger, and M. Sig. Scheduling transportation of live animals to avoid the spread of diseases. *Transportation Science*, 38:197–209, 2004.

M. Sol. *Column generation techniques for pickup and delivery problems*. PhD thesis, Technische Universitet Eindhoven, 1994.

M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

P. Toth and D. Vigo. *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, Philadelphia, 2002.

L. A. Wolsey. *Integer Programming*. Wiley-Interscience series in discrete mathematics. Wiley-Interscience publication, 1998.

H. Xu, Z.-L. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37:347–364, 2003.

Name	$\underline{z}$	Cost	Time	Name	$\underline{z}$	Cost	Time
AA30	**31119.1**	**31119.1**	75	LR1_2_1	**4819.1**	**4819.1**	174
AA35	**31299.8**	**31299.8**	93	LR1_2_2	**4093.1**	**4093.1**	212
AA40	**31515.9**	**31515.9**	62	LR1_2_3	3484.1	3486.8	264
AA45	**31759.8**	**31759.8**	124	LR1_2_4		2830.7	411
AA50	**41775.0**	**41775.0**	159	LR1_2_5	**4221.6**	**4221.6**	182
AA55	**41907.8**	**41907.8**	185	LR1_2_6	**3763.0**	**3763.0**	238
AA60	**42140.7**	**42140.7**	118	LR1_2_7		3112.9	272
AA65	**42250.2**	42252.7	135	LR1_2_8		2645.4	383
AA70	**42452.3**	42455.0	187	LR1_2_9	**3953.5**	**3953.5**	193
AA75	43206.5	52472.7	308	LR1_2_10	3376.2	3389.2	225
BB30	**31086.3**	**31086.3**	76	LC1_2_1	**2704.6**	**2704.6**	183
BB35	**31281.2**	**31281.2**	92	LC1_2_2	**2764.6**	**2764.6**	206
BB40	**31493.4**	**31493.4**	63	LC1_2_3	**2772.2**	**2772.2**	230
BB45	**41555.1**	**41555.1**	127	LC1_2_4		2661.4	341
BB50	**41701.0**	41703.4	160	LC1_2_5	**2702.0**	**2702.0**	203
BB55	**41885.7**	**41885.7**	96	LC1_2_6	**2701.0**	**2701.0**	204
BB60	**62420.1**	**62420.1**	178	LC1_2_7	**2701.0**	**2701.0**	221
BB65	**62639.1**	**62639.1**	202	LC1_2_8	**2689.8**	**2689.8**	221
BB70	**62951.0**	62952.3	236	LC1_2_9	**2724.2**	**2724.2**	230
BB75	62232.6	63130.4	256	LC1_2_10	2734.9	2741.6	260
CC30	**31087.7**	**31087.8**	76	LRC1_2_1	**3606.1**	**3606.1**	190
CC35	**31230.6**	**31230.6**	97	LRC1_2_2	**3292.4**	**3292.4**	208
CC40	**31358.5**	**31358.5**	132	LRC1_2_3		3079.5	271
CC45	**31509.1**	**31509.1**	82	LRC1_2_4		2525.8	431
CC50	**41685.3**	41689.0	168	LRC1_2_5	**3715.8**	**3715.8**	208
CC55	**41836.3**	**41836.3**	196	LRC1_2_6	**3360.9**	**3360.9**	198
CC60	37839.7	42015.5	127	LRC1_2_7	3298.2	3317.7	212
CC65	39480.3	42172.1	145	LRC1_2_8	3025.8	3086.7	220
CC70	42124.5	52201.9	288	LRC1_2_9	2995.5	3058.5	229
CC75	43565.0	52375.6	325	LRC1_2_10		2837.5	248
DD30	**21133.3**	**21133.3**	49	LR1101	**56744.9**	56791.7	2278
DD35	**31210.9**	31224.0	99	LR1105	52401.2	52901.3	2403
DD40	**31352.2**	**31352.2**	136	LC1101	**42488.7**	**42488.7**	1870
DD45	**31483.9**	**31484.0**	132	LC1105	**42477.4**	**42477.4**	1880
DD50	**31600.9**	**31600.9**	105	LRC1101	48198.7	48666.4	2145
DD55	**31743.3**	**31743.3**	124	LRC1105		49287.0	2468
DD60	31466.6	41869.4	247				
DD65	35313.7	42125.7	209				
DD70	36690.6	42220.3	175				
DD75	38762.1	42396.8	201				

Table 2: Results from PDPTW Heuristic

	SP1					SP2						
	No cuts	IPC	CC	2PC	Full	No cuts	IPC	CC	2PC	FC	RC	Full
AA30	84.10	0.00	0.00	0.17	0.17	84.10	0.00	0.00	0.17	0.00	0.00	0.17
AA35	84.22	0.00	1.10	1.43	1.42	84.22	0.00	1.02	1.34	0.00	0.00	1.34
AA40	99.97	0.00	100.00	88.32	100.00	99.97	0.00	91.22	80.72	0.00	0.00	93.35
AA45	100.00					100.00						
AA50	81.61	0.00	0.05	0.20	0.20	81.58	0.03	0.05	0.18	0.13	0.14	0.32
AA55	88.26	0.00	0.04	0.08	0.08	88.25	0.00	0.03	0.12	0.13	0.14	0.21
AA60	92.28	0.00	0.13	0.14	0.14	92.20	0.00	0.07	0.12	0.88	0.88	0.97
AA65	94.71	0.00	0.16	0.27	0.16	94.14	4.86	0.92	5.39	5.63	6.49	6.70
AA70	97.14	0.00	0.44	1.58	1.87	96.94	0.96	0.51	4.76	5.04	5.04	6.44
AA75	82.26	0.00	0.09	0.43	0.46	81.91	0.68	0.79	1.17	0.92	0.92	1.58
BB30	72.79	0.00	0.00	1.71	1.71	72.58	0.00	0.00	1.76	0.06	0.01	1.77
BB35	86.50	0.00	0.00	0.49	0.49	86.32	0.00	0.00	1.59	0.06	1.28	1.77
BB40	96.22	0.00	0.00	0.03	0.03	96.18	0.00	0.00	0.24	0.26	0.28	0.57
BB45	78.82	0.00	0.00	0.02	0.02	78.55	0.00	0.00	0.30	0.00	0.00	0.30
BB50	86.16	0.00	0.00	0.04	0.04	86.12	0.00	0.00	0.00	0.03	0.23	0.26
BB55	94.98	0.00	0.00	0.00	0.00	94.67	0.00	0.00	0.03	0.07	0.00	0.08
BB60	87.99	0.00	0.00	0.00	0.01	87.99	0.00	0.00	0.00	0.00	0.00	0.00
BB65	89.26	0.00	0.00	0.00	0.01	89.26	0.00	0.00	0.00	0.00	0.00	0.01
BB70	97.75	0.00	0.01	0.32	0.34	97.75	0.00	0.01	0.33	0.00	0.00	0.35
BB75	98.58	0.00	0.00	0.30	0.31	98.58	0.00	0.00	0.30	0.00	0.00	0.30
CC30	73.37	0.00	0.00	0.09	0.09	72.54	0.00	0.00	0.43	0.00	0.00	0.44
CC35	77.64	0.00	0.00	0.14	0.14	77.09	0.00	0.00	0.14	0.06	0.00	0.17
CC40	80.65	0.00	0.00	0.07	0.07	80.41	0.00	0.00	0.13	0.00	0.00	0.18
CC45	91.84	0.00	0.00	0.00	0.00	91.41	0.00	0.00	0.00	0.05	0.62	0.72
CC50	81.70	0.00	0.00	0.00	0.00	81.66	0.00	0.00	0.03	0.00	0.00	0.02
CC55	87.07	0.00	0.08	0.14	0.14	87.03	0.00	0.04	0.05	0.05	0.00	0.10
CC60	90.06	0.00	0.02	0.01	0.02	89.89	0.00	0.00	0.00	0.00	0.00	0.00
CC65	93.62	0.00	0.00	0.00	0.00	93.38	0.00	0.26	0.29	0.00	0.24	0.40
CC70	80.68	0.00	0.08	0.01	0.08	80.50	0.00	0.04	0.05	0.00	0.03	0.09
CC75	83.17	0.00	0.03	0.03	0.03	82.96	0.00	0.05	0.05	0.00	0.00	0.05
DD30	88.50	0.00	0.03	99.78	100.00	87.92	0.00	0.00	99.57	0.00	0.00	99.85
DD35	68.97	0.00	0.00	0.98	0.98	68.43	0.00	0.00	1.28	0.17	0.07	1.38
DD40	73.80	0.00	0.00	0.18	0.18	73.40	0.00	0.00	0.06	0.06	0.06	0.18
DD45	79.00	0.00	0.13	0.10	0.13	78.75	0.00	0.05	0.13	0.15	0.15	0.29
DD50	84.13	0.00	0.09	0.11	0.13	83.83	0.00	0.48	0.03	0.00	0.00	0.29
DD55	90.83	0.00	0.07	0.10	0.18	90.66	0.00	0.11	0.15	0.00	0.00	0.17
DD60	75.13	0.00	0.03	0.08	0.08	74.94	0.00	0.11	0.11	0.01	0.00	0.12
DD65	83.83	0.00	0.01	0.01	0.01	83.50	0.00	0.00	0.02	0.23	0.22	0.23
DD70	86.90	0.00	0.00	0.00	0.00	86.58	0.35	0.00	0.75	1.15	1.30	1.34
DD75	91.42	0.00	0.00	0.07	0.07	91.04	0.00	0.01	0.11	0.66	0.86	1.00
Avg.	86.40	0.00	2.63	5.06	5.38	86.18	0.18	2.46	5.18	0.41	0.49	5.73

Table 3: Impact of valid inequalities on data set one.

	SP1					SP2						
	No cuts	IPC	CC	2PC	Full	No cuts	IPC	CC	2PC	FC	RC	Full
LR1_2_5	99.98	100.00	0.00	0.00	100.00	99.98	100.00	0.00	0.00	0.00	0.00	100.00
LR1_2_6	100.00					99.98	68.85	0.00	100.00	100.00	0.00	100.00
LR1_2_9	99.34	2.45	0.00	72.65	72.81	99.34	2.45	0.00	68.97	0.00	0.00	71.81
LR1_2_10	99.61	0.00	0.00	2.80	2.80	99.43	0.00	0.00	8.83	2.50	2.50	10.35
LC1_2_9	99.63	0.00	0.00	14.15	14.03	99.54	0.00	0.00	2.92	0.00	0.00	2.92
LC1_2_10	99.73	0.00	0.00	10.10	10.10	99.61	0.00	11.07	37.41	3.76	19.03	38.41
LRC1_2_1	99.92	13.27	0.00	100.00	100.00	99.92	13.27	0.00	100.00	0.00	0.00	100.00
LRC1_2_2	99.14	0.00	0.00	80.03	80.03	99.14	0.00	0.00	80.03	0.00	0.00	80.03
LRC1_2_5	99.84	14.04	0.00	16.36	16.36	99.61	12.81	0.00	49.76	3.18	6.15	56.66
LRC1_2_7	99.31	0.00	0.00	14.22	14.57	98.88	0.00	6.32	8.66	7.38	15.05	22.30
LRC1_2_8	97.98	0.00	0.00	15.16	14.94	97.65	0.00	0.00	15.85	0.12	0.04	15.90
LRC1_2_9	97.51	0.00	0.00	17.03	17.03	96.97	0.36	0.48	15.28	0.72	0.97	16.05
Avg.	99.33	10.81	0.00	28.54	36.89	99.17	16.48	1.49	40.64	9.80	3.64	51.20

Table 4: Impact of valid inequalities on instances proposed by Li and Lim (2001).

		B&C		SP1				SP2			
Name	$\bar{z}$	$\underline{z}$	Time	$\underline{z}$	Time	Nodes	Cuts	$\underline{z}$	Time	Nodes	Cuts
AA30	31119.1	24513.3	6.0	26179.4	6.5	3	7	26179.4	11.2	3	25
AA35	31299.8	26337.5	9.1	26431.6	15.9	5	16	26427.5	34.4	5	25
AA40	31515.9	31501.5	19.3	31515.9	13.4	1	9	31515.2	47.1	5	28
AA45	31759.8	31701.0	1484.4	31759.8	15.9	1	0	31759.8	25.5	1	0
AA50	41775.0	31843.4	832.1	34108.5	72.0	7	11	34105.5	139.6	5	38
AA55	41907.8	33064.2	65.7	36992.7	65.6	3	11	36992.3	125.8	3	23
AA60	42140.7	33351.2	234.2	38892.1	235.7	5	6	38887.5	256.5	5	30
AA65	42250.2	32547.2	519.7	40020.9	349.5	9	5	39940.9	599.7	13	36
AA70	42452.3	32587.7		41262.0	2477.5	75	34	41237.5		160	78
AA75	52472.7	32599.4		43206.5		188	47	43131.4		97	85
BB30	31086.3	21129.0	63.4	22772.9	6.0	3	5	22712.0	9.0	3	8
BB35	31281.2	23659.7	13.9	27078.4	18.0	5	9	27076.2	33.2	7	16
BB40	31493.4	23782.1	32.6	30304.1	18.9	3	6	30298.5	35.8	3	17
BB45	41555.1	23891.8		32754.8	46.3	9	4	32669.7	58.2	7	9
BB50	41701.0	29072.1	482.6	35930.7	192.6	25	14	35929.6	159.9	11	16
BB55	41885.7	27863.2		39781.5	55.8	3	0	39656.8	85.5	3	3
BB60	62420.1	42605.8		54925.9	181.7	27	7	54925.5	276.7	35	6
BB65	62639.1	42901.3		55910.9	294.8	25	4	55910.9	486.2	33	6
BB70	62951.0	44114.8		61537.7	1839.4	139	15	61537.8	2182.8	117	33
BB75	63127.5	44491.3		62232.6		265	34	62232.5		200	45
CC30	31087.7	11134.5		22817.1	11.3	5	1	22588.7	19.2	5	8
CC35	31230.6	11277.9		24257.8	59.1	17	13	24087.3	80.9	11	18
CC40	31358.5	11373.3		25293.8	254.8	29	15	25224.8	658.7	39	74
CC45	31509.1	11514.1		28939.1	175.9	11	5	28822.2	843.8	27	58
CC50	41685.3	11690.2		34058.3	1962.3	137	16	34041.7	4005.2	145	98
CC55	41836.3	11812.0		36432.5	2729.2	117	15	36414.1		180	75
CC60	42015.5	11976.2		37839.7		162	22	37767.7		113	51
CC65	42172.1	12094.7		39480.3		85	21	39391.5		59	33
CC70	52201.9	12198.3		42124.5		44	12	42029.2		36	37
CC75	52375.6	12340.2		43565.0		47	7	43456.6		26	34
DD30	21133.3	11117.6		21133.3	25.2	1	25	21129.5	75.5	5	25
DD35	31210.9	11212.6		21620.2	765.1	143	38	21494.5	2266.8	261	128
DD40	31352.2	11324.1		23152.6	160.8	15	11	23028.4	687.5	37	47
DD45	31483.9	11433.9		24880.6	525.6	31	29	24811.6	1313.3	41	64
DD50	31600.9	11525.2		26593.7	1976.0	77	47	26506.4	4238.8	115	84
DD55	31743.3	11600.6		28836.5	1178.5	25	10	28782.0	3951.6	63	39
DD60	41869.4	11724.0		31466.6		88	10	31389.2		66	44
DD65	42125.7	12018.9		35313.7		59	22	35191.3		42	60
DD70	42220.3	12094.3		36690.6		55	10	36630.7		31	61
DD75	42396.8	12245.1		38762.1		42	13	38635.8		24	65
	#Solved		12		30				28		

Table 5: Branch-and-cut-and-price results on the first data set.

		B&C		SP1				SP2			
Name	$\bar{z}$	$\underline{z}$	Time	$\underline{z}$	Time	Nodes	Cuts	$\underline{z}$	Time	Nodes	Cuts
LR1_2_1	4819.1	4819.1	287.5	4819.1	5.1	1	0	4819.1	112.4	1	0
LR1_2_2	4093.1	4067.4		4093.1	84.0	1	0	4093.1	214.1	1	0
LR1_2_3	3486.9	3312.0						3484.1		2	1
LR1_2_4	2830.7	2452.8									
LR1_2_5	4221.6	4215.0	1075.5	4221.6	11.7	1	1	4221.6	130.6	1	1
LR1_2_6	3763.0	3482.1		3763.0	1041.6	1	0	3763.0	2198.9	1	2
LR1_2_7	3112.9	2773.2									
LR1_2_8	2645.5	2149.3									
LR1_2_9	3953.5	3815.1		3946.4	418.5	25	29	3946.1	2503.9	93	153
LR1_2_10	3389.2	2879.5		3376.2		4	3				
LC1_2_1	2704.6	2704.6	180.3	2704.6	4.9	1	0	2704.6	116.5	1	0
LC1_2_2	2764.6	2753.8	4979.5	2764.6	27.6	1	0	2764.6	140.2	1	0
LC1_2_3	2772.2	2561.2		2772.2	250.1	1	0	2772.2	240.6	1	0
LC1_2_4	2661.4	1944.6									
LC1_2_5	2702.0	2702.0	223.9	2702.0	6.3	1	0	2702.0	89.5	1	0
LC1_2_6	2701.0	2701.0	580.9	2701.0	9.8	1	0	2701.0	96.1	1	0
LC1_2_7	2701.0	2701.0	423.0	2701.0	10.9	1	0	2701.0	100.1	1	0
LC1_2_8	2689.8	2316.8		2689.8	31.5	1	0	2689.8	118.6	1	0
LC1_2_9	2724.2	1966.8		2715.6	6628.6	91	11	2712.1		73	18
LC1_2_10	2741.6	1493.7		2734.9		9	5	2734.9		9	52
LRC1_2_1	3606.1	3569.1	2485.5	3606.1	12.6	1	3	3606.1	154.1	1	3
LRC1_2_2	3292.4	3026.6		3286.8	1440.4	3	23	3286.8	1053.3	3	21
LRC1_2_3	3079.5	2529.8									
LRC1_2_4	2525.8	2103.1									
LRC1_2_5	3715.8	3333.6		3710.9	517.8	17	7	3709.6	1419.8	27	99
LRC1_2_6	3360.9	3072.7		3360.9	27.7	1	0	3360.9	142.4	1	2
LRC1_2_7	3317.7	2720.4		3298.2		81	25	3289.0		40	629
LRC1_2_8	3086.7	2441.6						3025.8		2	277
LRC1_2_9	3058.6	2261.3		2995.5		2	21	2980.7		2	50
LRC1_2_10	2837.5	2074.2									
	Avg.		1279.5		619.3				552.0		
	Solved	8		17				16			

Table 6: Branch-and-cut-and-price results on the second data set (instances proposed by Li and Lim (2001)). 100 requests.

		SP1				SP2			
name	UB	$\underline{z}$	time	nodes	Cuts	$\underline{z}$	time	nodes	Cuts
LR1101	56744.9	56740.9	1682.3	3	0	56740.9	2514.8	3	0
LR1105	52901.3	52401.2		8	34	52399.9		8	49
LC1101	42488.7	42488.7	778.9	1	0	42488.7	704.8	1	0
LC1105	42477.4	42477.4	762.7	1	0	42477.4	940.9	1	0
LRC1101	48666.5	48198.7		9	59	48192.7		7	89
LRC1105	49287.1								
	solved	3				3			

Table 7: Branch-and-cut-and-price results on the second data set (instances proposed by Li and Lim (2001)). Instances with 500 requests and tight time windows.

	Root lower bound				Time (s)				B&B Nodes			
	SP1	SP1	SP2	SP2	SP1	SP1	SP2	SP2	SP1	SP1	SP2	SP2
	+cuts	-cuts	+cuts	-cuts	+cuts	-cuts	+cuts	-cuts	+cuts	-cuts	+cuts	-cuts
CC45	91.84	91.84	91.47	91.41	176	221	845	1150	11	15	27	63
DD45	79.03	79.00	78.81	78.75	526	535	1313	2073	31	41	41	113
XX45	79.98	79.93	79.45	79.01	677	375	1022	1162	53	29	51	69
YY45	90.34	90.34	90.00	89.99					8	8	12	12
LR1_2_6	100.00	100.00	100.00	99.98	1072.9	1055.4	2230.1	4341.9	1	1	1	3
LR1_2_9	99.82	99.34	99.81	99.34	415		2496		25	1009	93	1141
LRC1_2_2	99.83	99.14	99.83	99.14	1433	2272	1048	1378	3	11	3	11
Avg.	91.55	91.37	91.34	91.09	777	892	1291	2021	19	159	33	202

Table 8: Impact of valid inequalities on a subset of instances.

Name	PP	Time total	Time preproc.	Time LP	Time PP heur.	Time PP exact	Time sep.	#Cuts	#Cols	#PP heur	#PP exact	#Labels
CC45	SP1	33.9	0.1	0.9	30.6	0.8	1.0	0	2306	140	4	90061
	SP2	96.8	6.4	1.3	85.9	0.4	1.9	13	2763	197	5	55492
DD45	SP1	41.8	0.1	1.0	37.3	1.3	1.4	4	2084	183	6	100036
	SP2	88.9	6.2	1.0	78.7	0.7	1.4	11	2192	199	5	84536
XX45	SP1	23.0	0.1	0.7	21.1	0.1	0.5	2	2094	128	1	29365
	SP2	83.3	5.4	1.2	73.8	0.1	1.8	14	2575	183	2	31878
YY45	SP1	843.3	0.1	1.5	93.2	747.3	0.4	0	2843	213	15	987681
	SP2	631.9	6.7	2.2	213.1	406.4	1.8	2	3242	319	22	670293
LR1_2_6	SP1	1085.0	0.3	0.5	75.4	996.6	5.4	0	1650	127	3	2905131
	SP2	2226.8	90.2	0.4	80.9	2021.9	26.8	2	1679	119	3	4595936
LR1_2_9	SP1	77.4	0.3	0.4	50.0	3.4	15.9	17	1553	142	1	199984
	SP2	367.1	55.3	0.4	66.7	5.6	231.2	21	1431	150	2	214272
LRC1_2_2	SP1	169.1	0.2	0.3	97.7	45.9	17.5	23	1591	152	2	729012
	SP2	681.1	77.1	0.4	151.4	64.8	377.6	21	1620	197	2	922721
Avg.	SP1	324.8	0.1	0.7	57.9	256.5	6.0	6.6	2017	155	5	720181
	SP2	596.6	35.3	1.0	107.2	357.1	91.8	12.0	2215	195	6	939304

Table 9: Detailed results for the root node on a subset of instances.

Name	Total time (s)				#calls SP heuristic				#calls SP Exact			
	SP1 std.	SP1 simple	SP2 std.	SP2 simple	SP1 std.	SP1 simple	SP2 std.	SP2 simple	SP1 std.	SP1 simple	SP2 std.	SP2 simple
CC45	33.9	20.1	96.8	21.5	140	232	197	282	4	64	5	82
DD45	41.8	24.2	88.9	24.9	183	367	199	414	6	64	5	83
XX45	23.0	22.7	83.3	32.4	128	497	183	601	1	58	2	60
YY45	843.3	2263.4	631.9	1078.2	213	445	319	459	15	54	22	66
LR1_2_6	1085.0		2226.8	6901.9	127	269	119	344	3	2	3	25
LR1_2_9	77.4	176.4	367.1	389.7	142	323	150	372	1	21	2	25
LRC1_2_2	169.1	2277.6	681.1	994.6	152	298	197	306	2	18	2	17
Avg.	198.1	797.4	324.9	423.5	160	360	208	406	5	47	6	56

Table 10: Comparison of pricing heuristics on a subset of instances.