CIRRELT

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

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Branch-and-Cut Algorithm for the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks

Jean-François Côté
Claudia Archetti
Maria Grazia Speranza
Michel Gendreau
Jean-Yves Potvin

December 2010

CIRRELT-2010-55

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
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Branch-and-Cut Algorithm for the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks

Jean-François Côté[1,2], Claudia Archetti[3], Maria Grazia Speranza[3], Michel Gendreau[1,4], Jean-Yves Potvin[1,2,*]

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

[2] Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-ville, Montréal, Canada H3C 3J7

[3] Dipartimento Metodi Quantitativi, Università degli Studi di Brescia, Contrada Santa Chiara 50, 25122 Brescia, Italia

[4] Department of Mathematics and Industrial Engineering, École Polytechnique de Montréal, P.O. Box 6079, Station Centre-ville, Montréal, Canada H3C 3A7

Abstract. This paper studies the pickup and delivery traveling salesman problem with multiple stacks. The vehicle contains a number of (horizontal) stacks of finite capacity for loading items from the rear of the vehicle. Each stack must satisfy the last-in-first-out constraint which states that any new item must be loaded on top of a stack and any unloaded item must be on top of its stack. A branch-and-cut algorithm is proposed for solving this problem. Computational results are reported on different types of randomly generated instances, as well as on classical instances for some well-known special cases of the problem.

Keywords. Traveling salesman, pickup, delivery, loading, multiple stacks.

Acknowledgements. Financial support for this work was provided by the Natural Sciences and Engineering Council of Canada (NSERC). This support is gratefully acknowledged. We also want to thank Dr. Manuel Iori for his helpful comments.

---

* Corresponding author: Jean-Yves.Potvin@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2010

1 Introduction

While the vehicle routing problem has been studied for a long time and hundreds of papers have dealt with solution algorithms for different variants of this problem, much less attention has been paid to problems with both routing and loading aspects. As vehicle routing problems are hard problems, integrating routing and loading is a challenging task. However, this is the kind of problems that carriers must face on a daily basis. By solving the routing and loading problems separately, grossly sub-optimal solutions are obtained. For example, if the loading problem is solved first, the optimization of the loading space is likely to lead to severe constraints on the routing aspect of the problem and to time consuming unloading and reloading operations along the route.

In this paper, we study the pickup and delivery traveling salesman problem with multiple stacks (*PDTSPMS*), where a single vehicle is available to serve a set of customer requests. Each request consists in picking up an item at a given location and delivering it to a different location. The vehicle contains a number of independent horizontal stacks (rows) of finite length (capacity) for loading items from the rear. Each stack must satisfy the last-in-first-out (LIFO) property which states that any new item must be loaded on top of a stack and any unloaded item must be on top of its stack. The objective is to serve all requests at minimum cost, where the cost corresponds to the traveling distance.

The *PDTSPMS* is a generalization of the double traveling salesman problem with multiple stacks (*DTSPMS*) introduced in [27]. In the *DTSPMS*, a number of LIFO stacks are available to stock items of equal length. In a valid solution, the vehicle must collect all pickups and return to the depot before performing the deliveries. In [27], the authors propose a mathematical model and some metaheuristics to solve the *DTSPMS*, while new neighborhood structures are considered in [14]. A large neighborhood search heuristic developed primarily for the *PDTSPMS* is also successfully applied to the *DTSPMS* in [10]. Exact solution methods are reported in [22, 26]: the second work is based on a branch-and-cut procedure, while the first one matches the k best tours for the pickups and the k best tours for the deliveries. An additive branch-and-bound method is proposed in [4] for a special case with only two stacks. Finally, some theoretical properties of the DTSPMS are derived in [6] and a heuristic algorithm is developed to exploit these properties.

The *PDTSPMS* is also a generalization of the pickup and delivery TSP with LIFO loading constraints (*PDTSPL*), where the vehicle contains a single LIFO stack of infinite capacity. This problem is solved with either heuristic of exact methods in [3, 5, 9]. A polyhedral study and a branch-and-cut algorithm are also described in [12]. Other variants of pickup and delivery traveling salesman problems with different types of loading policies, like first-in-first-out (FIFO), are reported in [2, 3, 8, 13].

Many types of combined vehicle routing and loading problems can be found in

the scientific literature when the routing problem is made of pickup- or (exclusive) delivery-only operations. A tabu search and an ant colony heuristic are used to solve a problem with two dimensional loading constraints in [15, 18], while an exact approach is proposed in [21]. The problem is extended to three dimensional loading constraints and is solved with metaheuristics in [16]. Other types of loading constraints have also been considered in [11, 17, 32]. For an overview of this line of research, the reader is referred to [20].

In this paper, several classes of valid inequalities for the *PDTSPMS* are derived and exploited within a branch-and-cut algorithm. The computational results show that the size of instances that can be solved with this algorithm strongly depends on the specific type of instance considered. While instances with up to 43 nodes can be solved in some test sets, instances with 27 nodes cannot be solved exactly in some other sets. Since the branch-and-cut algorithm proposed in this work is the first exact algorithm for the *PDTSPMS*, *DTSPMS* instances have been used to compare its performance with other exact methods, namely the algorithms of Lusby et al. [26] and Petersen et al. [22], for a variable number of stacks, and the algorithm of Carrabs et al. [4] for two stacks. The results show that our algorithm outperforms these previous methods, even if it was not specifically designed for the *DTSPMS*.

The paper is organized as follows. In Section 2, three different formulations for the *PDTSPMS* are proposed. Then, several classes of valid inequalities are presented in Section 3 and separation procedures are described in Section 4. Section 5 is devoted to the branch-and-cut algorithm and computational results are reported in Section 6.

2 Problem formulation

The *PDTSPMS* can be formally stated as follows. Let $G = (V, A)$ be a complete directed graph where $V = \{0, 1, ..., 2n+1\}$ is the node set and A is the arc set. Nodes 0 and $2n + 1$ denote the depot at the start and at the end of the tour, respectively, while nodes i and $n+i$ are the pickup and delivery locations of customer i, $1 \leq i \leq n$. Each request implies to pickup an item at location i and to deliver it at location $n+i$. We denote $P = \{1, ..., n\}$ and $D = \{n+1, ..., 2n\}$ the set of pickup and delivery locations, respectively. An item of length d_i is associated with pickup location $i \in P$. For simplicity, we will refer in the following to the item picked up at location i as item i. An item of length $d_0 = d_{2n+1} = 0$ is associated with the depot and an item of length $-d_i$ with delivery location $n + i \in D$. We also have a cost c_{ij} associated with each arc $(i, j) \in A$. A single vehicle is available to serve all requests. This vehicle contains a set $M = \{1, 2, ..., m\}$ of LIFO stacks, each of capacity Q, to transport the items between pickup and delivery locations. The goal is to serve all customers with a least-cost route, starting at node 0 and ending at node $2n + 1$, while satisfying the side constraints.

To model the problem as a mathematical program, we introduce the following notation:

- $\bar{S} = V \backslash S$, $S \subseteq V$;
- $x(S) = \sum_{i,j \in S} x_{ij}$, $S \subseteq V$;
- $x(\delta^+(S)) = \sum_{i \in S, j \notin S} x_{ij}$, $S \subseteq V$;
- $x(\delta^-(S)) = \sum_{i \notin S, j \in S} x_{ij}$, $S \subseteq V$;
- $x(\delta(S)) = x(\delta^+(S)) + x(\delta^-(S))$, $S \subseteq V$;
- $x(i,S) = \sum_{j \in S} x_{ij},\ i \in V$, $S \subseteq V$;
- $x(S,i) = \sum_{j \in S} x_{ji},\ i \in V$, $S \subseteq V$;
- $\pi(j) = j,\ j \in P$;
- $\pi(n+j) = j,\ n+j \in D$;
- $\pi(S) = \{i \in P | n+i \in S\}$;
- $\sigma(S) = \{n+i \in D | i \in S\}$;
- $S_j = \{S' \subset P \cup D | j \in S' \text{ and } n+j \notin S'\}$;
- $S_{n+j} = \{S' \subset P \cup D | j \notin S' \text{ and } n+j \in S'\}$;

We also define the following decision variables:

- x_{ij} is 1 if node j is visited immediately after node i, 0 otherwise, $i,j \in V$, $i \neq j$;
- y_{ik} is 1 if item i is loaded in stack k, 0 otherwise, $i \in P$, $k \in M$;
- $0 \leq s_{ik} \leq Q$ is the load of stack k upon leaving node i, $i \in V$, $k \in M$ (with $s_{0k} = 0$, $k \in M$).

The *PDTSPMS* can now be formulated as follows:

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{j \in V} x_{ij} = 1, \quad i \in P \cup D \cup \{0\} \tag{2}$$

$$\sum_{j \in V} x_{ji} = 1, \quad i \in P \cup D \cup \{2n+1\} \tag{3}$$

$$x(S) \leq |S| - 1, \quad S \subset P \cup D, |S| \geq 2 \tag{4}$$

$$x(S) \leq |S| - 2, \quad S \in \Omega \tag{5}$$

$$\sum_{k \in M} y_{ik} = 1, \quad i \in P \tag{6}$$

$$s_{jk} \geq s_{ik} + d_j y_{\pi(j)k} - Q(1 - x_{ij}), \quad i \in V,\ j \in P \cup D,\ k \in M \tag{7}$$

$$s_{jk} \leq s_{ik} + d_j y_{\pi(j)k} + Q(1 - x_{ij}), \quad i \in V,\ j \in P \cup D,\ k \in M \tag{8}$$

$$s_{(n+j)k} \geq s_{jk} - d_j y_{jk} - Q(1 - y_{jk}), \quad j \in P,\ k \in M \tag{9}$$

$$s_{(n+j)k} \leq s_{jk} - d_j y_{jk} + Q(1 - y_{jk}), \quad j \in P,\ k \in M \tag{10}$$

$$s_{0k} = 0, \quad k \in M \tag{11}$$

$$0 \leq s_{ik} \leq Q, \quad i \in V,\ k \in M \tag{12}$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in V \tag{13}$$

$$y_{ik} \in \{0, 1\}, \quad i \in P,\ k \in M. \tag{14}$$

This model will be referred to as the *network formulation*. In this formulation, the objective function (1) is aimed at minimizing the total cost which corresponds to the distance traveled by the vehicle. Each node is visited exactly once through constraints (2) and (3). Constraints (4) impose connectivity on the route. The precedence constraints between the pickup and delivery locations (5) are taken from [30]. Here, the set Ω is a collection of subsets $S \subset V$ such that, for each subset S, we have $0 \in S, 2n+1 \notin S$ and there exists a pickup $i \notin S$ for which $n+i \in S$. Constraints (6) state that each item is loaded in exactly one stack. Constraints (7) and (8) define the status of the stacks after each pickup and delivery. The LIFO loading constraints are stated in (9) and (10). Finally, constraints (11) and (12) take into account the capacity of each stack.

In the next sections, alternative formulations are considered to avoid the use of (7), (8), (9) and (10) which are similar to the Miller, Tucker and Zemlin constraints [25]. These constraints are known to generate poor linear relaxations.

2.1 Flow formulation

We can extend the flow formulation proposed in [9] for the *PDTSPL* by adding a stack index to the variables. Let f_{ij}^k be the flow circulating on arc (i, j) for stack k. A pickup operation increases the flow whereas a delivery operation decreases it. Thus, the flow on arc (i, j) for stack k increases by d_i if item i is loaded on stack k, as indicated in constraints (15) below. When item i is not loaded on stack k, the flow remains unchanged.

To satisfy the LIFO constraints, the load on stack k before serving $i \in P$ and after serving $n+i \in D$ must be the same. Constraints (16) and (17) guarantee the LIFO policy only for items loaded on the same stack. If item i is on stack k, inequalities (16) and (17) state that the flow to $i \in P$ and the flow from $n+i \in D$ must be the same. On the other hand, if item i is not on stack k, constraints (16) and (17) are not binding. The flow formulation is then given by (1)-(6), (13)-(14) and (15)-(18).

$$\sum_{j \in N} f_{ji}^k - \sum_{j \in N} f_{ij}^k = d_i y_{\pi(i)k} \quad i \in P \cup D \tag{15}$$

$$\sum_{j \in N} f_{ji}^k - \sum_{j \in N} f_{(n+i)j}^k \leq Q(1 - y_{ik}) \quad i \in P,\ k \in M \tag{16}$$

$$\sum_{j \in N} f_{ji}^k - \sum_{j \in N} f_{(n+i)j}^k \geq -Q(1 - y_{ik}) \quad i \in P,\ k \in M \tag{17}$$

$$0 \leq f_{ji}^k \leq Q \quad (i,j) \in A,\ k \in M. \tag{18}$$

2.2 Infeasible path formulation

Let p be a path in G. The formulation presented here is inspired by the work on the DTSPMS in [26] and requires no additional variables with regard to the network formulation. The infeasible path formulation is given by (1)-(5), (13) plus a set of constraints for eliminating all paths which are infeasible with regard to the LIFO constraints (9)-(10) or capacity constraints (12). These so-called *path inequalities* are of the form:

$$x(p) \leq |p| - 1 \quad p \in \Phi \tag{19}$$

where $x(p)$ is the sum, over all arcs in path p, of the x variables associated with these arcs, $|p|$ is the number of arcs in path p and Φ denotes the set of all LIFO or capacity infeasible paths.

As the number of LIFO or capacity infeasible paths can be huge, inserting a cut for each infeasible path is inefficient. Thus, we relax these constraints and check for possible violations to the LIFO or capacity constraints by solving a packing problem (see below) whenever a valid pickup and delivery traveling salesman tour is found. If the packing problem is infeasible, we add a constraint of type (19) to remove this path from the solution space. Conversely, if a feasible packing is found, we have a feasible solution to the *PDTSPMS*.

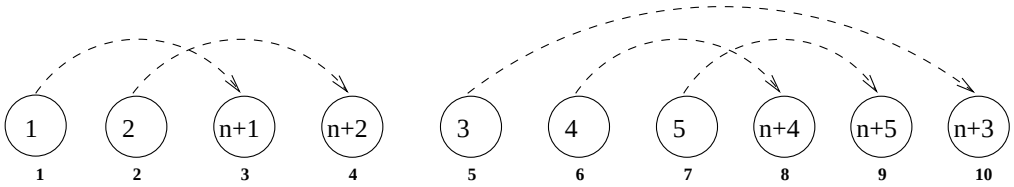

Figure 1: Example of a pickup and delivery sequence

2.2.1 Packing problem

Let p be a path. We want to check whether p is LIFO or capacity infeasible. From p it is possible to compute a set I of incompatible pairs of items (i, j), namely those items which cannot be packed on the same stack due to the LIFO constraints. This computation can be done in $O(n^2)$ by looking at each pair of nodes $i, j \in P$ visited in p, for which $i < j < n + i < n + j$ or $j < i < n + j < n + i$, where $i < j$ means that i is visited before j in p. For a set of items $\{i_1, i_2, ..., i_k\}$ for which $i_1 < i_2 < ... < i_k < n + i_1 < n + i_2 < ... < n + i_k$, we say that items $i_1, i_2, ..., i_k$ cross each other. We denote by O the list of nodes in the path sequenced by their visit order. Let also a_{io} be equal to 1 if item i is in the vehicle when node $o \in O$ is visited, 0 otherwise. Hence, $a_{io} = 1$ if item i is picked up before o and delivered after o (i.e., $i < o < n + i$).

In Figure 1, an example for a path with 5 pickups and 5 deliveries is depicted. The list O is equal to $\{1, 2, n + 1, n + 2, 3, 4, 5, n + 4, n + 5, n + 3\}$. With regard to incompatibilities, items 1 and 2 cross each other and cannot be on the same stack. The same applies to items 4 and 5. The set of incompatible pairs is then $I = \{(1, 2), (4, 5)\}$.

The objective is to find a feasible packing for the route, that is, an assignment of each item to a stack such that the LIFO and capacity constraints are satisfied. Let z_{ik} be a binary variable equal to 1 if item i is loaded on stack k. The packing problem is then:

$$\sum_{k \in M} z_{ik} = 1 \quad i \in P \tag{20}$$

$$z_{ik} + z_{jk} \leq 1 \quad (i, j) \in I \tag{21}$$

$$\sum_{i \in P} a_{io} d_i z_{ik} \leq Q \quad k \in M, \; o \in O \tag{22}$$

$$z_{ik} \in \{0, 1\} \quad i \in P, k \in M \tag{23}$$

In this formulation, constraints (20) associate a single stack with each item. Constraints (21) prohibit items on the same stack to cross each other. Capacity

constraints (22) state that the sum of the lengths of items which are on the same stack at the same time must be less than or equal to the capacity of the stack.

We conclude this section by noting that a similar problem arises in the *DTSPMS* (where deliveries can only be done once all items have been picked up). The authors in [6, 31] note that the packing problem in the *DTSPMS* is NP-hard, as it is equivalent to a Q-bounded graph coloring problem. However, they showed that the problem is polynomially solvable when there is no capacity constraint. Since our packing problem is much more complex, due to deliveries which are allowed to precede pickups, it is very unlikely that a polynomial time algorithm exists.

3 Valid inequalities

This section presents valid inequalities for the *PDTSPMS*. We first present the inequalities that are inherited from the classical pickup and delivery traveling salesman problem (*PDTSP*), where the loading aspect is not considered. Then, we introduce new inequalities obtained by adapting some inequalities initially proposed for the vehicle routing problem (*VRP*) and the *PDTSPL* .

3.1 Inequalities for the PDTSPL

Given that the *PDTSPMS* extends the *PDTSP*, all known inequalities for this problem can be used. Relevant work for the *PDTSP* can be found in [7, 12, 28, 29]. Here, we use the set of inequalities in [9], as described below.

(a) A first class of inequalities is obtained through the predecessor and successor inequalities for the precedence-constrained $ATSP$ [1]:

$$x(S) + \sum_{i \in S} \sum_{j \in \bar{S} \cap \pi(S)} x_{ij} + \sum_{i \in S \cap \pi(S)} \sum_{j \in \bar{S} \setminus \pi(S)} x_{ij} \leq |S| - 1 \quad S \subseteq P \cup D \tag{24}$$

$$x(S) + \sum_{i \in \bar{S} \cap \sigma(S)} \sum_{j \in S} x_{ij} + \sum_{i \in \bar{S} \setminus \sigma(S)} \sum_{j \in S \cap \sigma(S)} x_{ij} \leq |S| - 1 \quad S \subseteq P \cup D \tag{25}$$

(b) For a given ordered set $S = \{i_1, i_2, ..., i_k\} \subseteq V$ with $k \geq 3$, the following cycle inequalities for the $ATSP$ have been proposed in [19]:

$$\sum_{h=1}^{k-1} x_{i_h, i_{h+1}} + x_{i_k, i_1} + 2 \sum_{h=2}^{k-1} x_{i_h, i_1} + \sum_{h=3}^{k-1} \sum_{l=2}^{h-1} x_{i_h, i_l} \leq k - 1 \tag{26}$$

$$\sum_{h=1}^{k-1} x_{i_h,i_{h+1}} + x_{i_k,i_1} + 2\sum_{h=3}^{k} x_{i_1,i_h} + \sum_{h=4}^{k}\sum_{l=3}^{h-1} x_{i_h,i_l} \leq k-1 \tag{27}$$

The previous inequalities can be strengthened by considering sets $\pi(S)$ and $\sigma(S)$ to obtain the cycle inequalities [7]:

$$\sum_{h=1}^{k-1} x_{i_h,i_{h+1}} + x_{i_k,i_1} + 2\sum_{h=2}^{k-1} x_{i_h,i_1} + \sum_{h=3}^{k-1}\sum_{l=2}^{h-1} x_{i_h,i_l} + \sum_{n+i_p \in \bar{S}\cap\sigma(S)} x_{n+i_p,i_1} \leq k-1 \tag{28}$$

$$\sum_{h=1}^{k-1} x_{i_h,i_{h+1}} + x_{i_k,i_1} + 2\sum_{h=3}^{k} x_{i_1,i_h} + \sum_{h=4}^{k}\sum_{l=3}^{h-1} x_{i_h,i_l} + \sum_{i_p \in \bar{S}\cap\pi(S)} x_{i_1,i_p} \leq k-1 \tag{29}$$

(c) Finally, let $U_1, ..., U_k \subset P \cup D$ be mutually disjoint subsets such that $i_1, ..., i_k \in P$ are customer requests for which $i_l, n+i_{l+1} \in U_l$ for $l = 1, ..., k$ (where $i_{k+1} = i_1$). The precedence cycle breaking inequalities, introduced for the precedence-constrained TSP in [1], are also valid for the $PDTSPMS$:

$$\sum_{l=1}^{k} x(U_l) \leq \sum_{l=1}^{k} |U_l| - k - 1 \tag{30}$$

3.2 LIFO inequalities

In [9], the authors present a class of inequalities for the $PDTSPL$ that satisfies the LIFO constraints. Note again that this problem corresponds to a $PDTSPMS$ with a single stack of infinite capacity. In the $PDTSPL$, items are not allowed to cross each other. To impose a LIFO policy, the set Γ is defined as a collection of sets $S \subset P \cup D$ such that there exist $j \in S$ and $n+j \notin S$ or $j \notin S$ and $n+j \in S$. In [9] the authors proved that inequalities of type (31) are sufficient to prohibit all LIFO infeasible solutions. A graphical representation is shown in Figure 2.

$$x(i,S) + x(S) + x(S,n+i) \leq |S| \quad S \in \Gamma, i, n+i \notin S, i \in P. \tag{31}$$

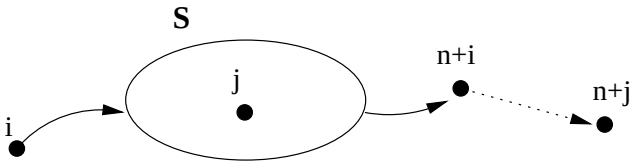

Figure 2: Forbidden pattern for LIFO constraint with one stack

Note that inequalities (31) are not valid for the *PDTSPMS* because multiple stacks are available. However, they can be adapted to the *PDTSPMS*. We first show how inequalities (31) can be extended to the *PDTSPMS* with two stacks. Then we derive a formulation for an arbitrary number of stacks.

In the *PDTSPMS*, items can be allowed to cross each other depending on the number of stacks. For example, consider a vehicle with two stacks. A situation where two items cross each other could be acceptable because the first item can be loaded on the first stack and the other on the second stack. However, a situation where three items cross each other is forbidden because three stacks would then be required. We conclude that for a vehicle with k stacks, a set of $k+1$ or more items cannot cross each other, otherwise an infeasible solution is obtained.

Let us show how constraints (31) can be extended to two stacks. We suppose that the LIFO constraint is violated and that the solution is feasible for the *PDTSP*. It means that we can find a path where three items cross each other. In other words, there exist $i, j, k \in P$ such that $i < j < k < n+i < n+j < n+k$. We can exclude this path by using a constraint of type (19). However, this approach is rather weak because inequality (19) cuts a single path. A different type of inequalities is proposed here based on an extension of (31). Let us define two subsets $S_j, S_k \subset P \cup D$ such that $j, k \in S_j; i, n+i, n+j, n+k \notin S_j$ and $k, n+i \in S_k; i, j, n+j, n+k \notin S_k$. A prohibited pattern is shown in Figure 3. By extending (31) to this pattern we obtain :

$$\begin{aligned} x(i, S_j) + x(S_j) + x(S_j, n+i) + \\ x(j, S_k) + x(S_k) + x(S_k, n+j) \leq \\ |S_j| + |S_k| + 1. \end{aligned} \tag{32}$$

Inequalities (32) forbid all paths from i to $n+i$ going through j and k (but not through $n+j$ and $n+k$) and, at the same time, forbid all paths from j to $n+j$ going through k and $n+i$ (but not through i and $n+k$).

Constraints (32) can be extended to impose LIFO constraints in the general case with an arbitrary number of stacks M. Let $\mathcal{L} = \{i_1, i_2, ..., i_{M+1} \in P, i_j \neq i_k\}$ be a set of pickup locations where the associated items mutually cross each other (i.e., each item in $\mathcal{L}$ crosses all other items in $\mathcal{L}$). This situation induces the following

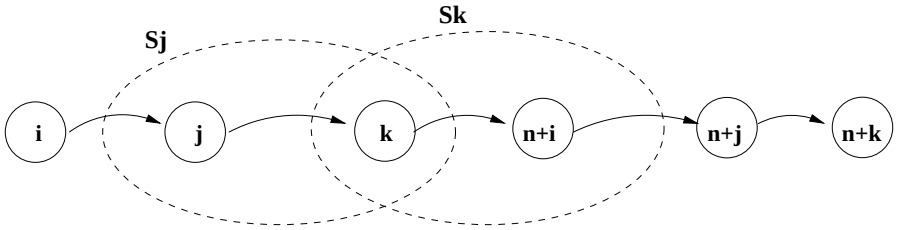

Figure 3: Forbidden pattern for LIFO constraint with two stacks

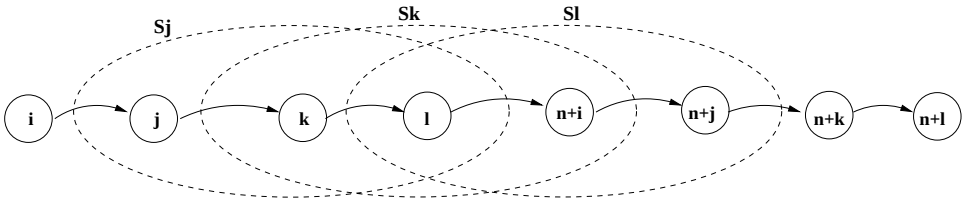

Figure 4: Forbidden pattern for LIFO constraint with three stacks

delivery pattern: $i_1 < i_2 < ... < i_{M+1} < n + i_1 < ... < n + i_{M+1}$. Also, let S_{i_h}, $2 \leq h \leq M + 1$, denote a set with nodes i_h to i_{M+1} and $n + i_1$ to $n + i_{h-2}$, but without nodes i_1 to i_{h-1} and $n + i_{h-1}$ to $n + i_{M+1}$. Note that S_{i_2} does not contain the delivery nodes $n + i_1$ to $n + i_{M+1}$. An example with three stacks is shown in Figure 4. Since all these requests cross each other, none of them can be on the same stack and at least $M + 1$ different stacks are required. Thus, we can exclude all paths with this delivery pattern by extending inequalities (32) in the following way:

$$\sum_{h=1}^{M}[x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n + i_h)] \leq \sum_{h=2}^{M+1} |S_{i_h}| + M - 1. \tag{33}$$

Using the line of reasoning for (32), inequalities (33) forbid all paths from i_h to $n + i_h$ going through nodes i_{h+1} to i_{M+1} and $n + i_1$ to $n + i_{h-1}$, for $h = 1, ..., M + 1$.

3.3 Capacity inequalities

In the classical capacitated *VRP*, the capacity constraints can be imposed as follow:

$$x(\delta(S)) \geq 2r(S) \quad S \subset P \cup D, \tag{34}$$

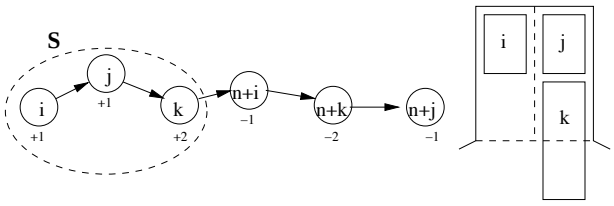

Figure 5: Violated capacity constraint

where $r(S)$ is the minimum number of vehicles required to serve all customers in S. However, computing $r(S)$ is difficult as it involves solving a bin packing problem. By replacing $r(S)$ with the lower bound $\lceil |\sum_{i \in S} d_i/Q| \rceil$, we obtain the *rounded capacity inequalities*

$$x(\delta(S)) \geq \lceil |\sum_{i \in S} d_i|/Q \rceil. \tag{35}$$

Inequalities (35) can be easily adapted to the *PDTSPMS* by ignoring the presence of multiple stacks. The whole loading area would then become available for a total capacity of MQ. The rounded capacity inequalities become:

$$x(\delta(S)) \geq 2\lceil |\sum_{i \in S} d_i|/(MQ) \rceil. \tag{36}$$

Note that we have $|\sum_{i \in S} d_i|$ in (36) because $d_i < 0$ for delivery locations. Unfortunately, these inequalities are not sufficient to cut all solutions that violate the capacity constraints. Consider the example shown in Figure 5 with two stacks and $Q = 2$. The total available capacity is equal to 4 and the sum of the lengths of the items in S is also equal to 4. Thus, (36) is satisfied. However, this solution is clearly infeasible because i and j cross each other. Thus, j and k must be in the same stack which is infeasible.

However, it is possible to derive a new set of inequalities for the capacity constraints. In the previous example, all items in conflict with i must fit in the remaining available capacity of the vehicle, which is $Q(M-1)$. By summing up the lengths of those items, we obtain a value of 3 which is larger than $Q(M-1)$, thus indicating that the solution is infeasible. The basic idea is presented in Figure 6, where D_1 is the sum of the lengths of items that are picked up before i and delivered before $n+i$, while P_1 is the sum of the lengths of items picked up between i and $n+i$ and delivered after $n+i$. Items corresponding to D_1 and P_1 cross i and cannot be on the same stack than i, which implies that they must fit in the $M-1$ remaining stacks.

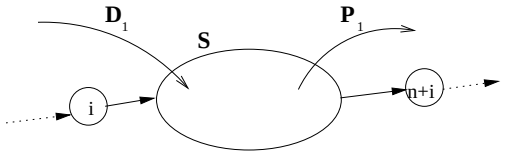

Figure 6: Conflict capacity constraint

To write these inequalities, we define $q(S) = \sum_{i \in S} d_i$ as the sum of the lengths of items in S and $z(S)$ as the maximum between the sum of the lengths of pickup and delivery items in S:

$$z(S) = \max\{q(\pi(S)\backslash S), -q(\sigma(S)\backslash S)\}. \tag{37}$$

Note that $\pi(S)\backslash S$ is the set of pickup nodes not in S and for which the corresponding delivery is in S, while $\sigma(S)\backslash S$ is the set of delivery nodes not in S and for which the corresponding pickup is in S.

When $z(S)$ is greater than $Q(M-1)$, then inequalities (38) can be added to forbid all paths from i to $n+i$ going through all nodes in S. We refer to this new type of inequalities as *conflict capacity inequalities*

$$x(i,S)+x(S)+x(S,n+i) \leq |S| \quad S \in P \cup D,\ i, n+i \notin S,\ z(S) > Q(M-1) \tag{38}$$

The previous inequalities can be extended by considering more items. Let $\mathcal{L} = \{i_1, i_2, ..., i_k \in P, 2 \leq k \leq M-1, i_j \neq i_k\}$ be a set where each item crosses all other items in the set. This situation induces the delivery pattern $i_1 < i_2 < ... < i_k < n+i_1 < ... < n+i_k$. Also, let S_{i_h}, $2 \leq h \leq k$, denote a set with nodes i_h to i_k and $n+i_1$ to $n+i_{h-2}$, but without nodes i_1 to i_{h-1} and $n+i_{h-1}$ to $n+i_k$. Note that S_{i_2} does not contain the delivery nodes $n+i_1$ to $n+i_{k+1}$. Since all these items cross each other, none of them can be on the same stack and at least k different stacks are required. Now, all the other items that cross items in $\mathcal{L}$ can only fit in the $M-k+1$ remaining stacks. These items are picked up before i_1 and delivered between i_k and $n+i_1$ or they are picked up between i_k and $n+i_1$ and delivered after $n+i_k$. The former case means that the deliveries associated with these items are in the intersection $S_{i_2} \cap ... \cap S_{i_k}$, while the pickups are not in the union $S_{i_2} \cup ... \cup S_{i_k}$. In the latter case, the pickups associated with these items are rather in the intersection $S_{i_2} \cap ... \cap S_{i_k}$ while the corresponding deliveries are not in the union $S_{i_2} \cup ... \cup S_{i_k}$. Denoting by $z(S_{i_2}, ..., S_{i_k})$ the maximum between the sum of the lengths of the items delivered and picked up in $S_{i_2} \cap ... \cap S_{i_k}$, we have:

$$\begin{aligned} z(S_{i_2}, ..., S_{i_k}) = \max\{&q(\pi(S_{i_2} \cap ... \cap S_{i_k})/(S_{i_2} \cup ... \cup S_{i_k})), \\ &-q(\sigma(S_{i_2} \cap ... \cap S_{i_k})/(S_{i_2} \cup ... \cup S_{i_k}))\}. \end{aligned} \tag{39}$$

If $z(S_{i_2}, ..., S_{i_k})$ exceeds the capacity $Q(M-k+1)$ of the remaining stacks, we have an infeasible solution which can be excluded through the following inequalities:

$$\sum_{h=1}^{k}[x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h)] \leq \sum_{h=2}^{k}|S_{i_h}| + k - 1, \quad \text{where } z(S_{i_2}, ..., S_{i_k}) > Q(M-k+1). \tag{40}$$

Inequalities (40) forbid all paths from i_h to $S_{i_{h+1}}$ and from $S_{i_{h+1}}$ to $n+i_h$ going through all nodes in $S_{i_{h+1}}$, $1 \leq h \leq k-1$. It is important to observe that it does not exclude all capacity constraint violations, because the sequence of pickups and deliveries in $S_{i_2} \cap ... \cap S_{i_k}$ is not known.

We can extend (40) to allow the *DTSPMS* to cover a broader range of capacity constraint violations. Given the particular structure of *DTSPMS* solutions, where all pickups are done before any delivery, we know that pick up items are served before delivery items in $S_{i_2} \cap ... \cap S_{i_k}$. Thus, all these items are in the vehicle at the same time. By summing up the demand of all pickup and delivery items in the intersection, we get:

$$z(S_{i_2}, ..., S_{i_k}) = q(\pi(S_{i_2} \cap ... \cap S_{i_k})/(S_{i_2} \cup ... \cup S_{i_k})) + q(\sigma(S_{i_2} \cap ... \cap S_{i_k})/(S_{i_2} \cup ... \cup S_{i_k})). \tag{41}$$

As before, if $z(S_{i_2}, ..., S_{i_k})$ exceeds the capacity $Q(M-k+1)$ of the remaining stacks, we have an infeasible solution for the *DTSPMS* which can be excluded through inequalities (40).

4 Separation procedures

4.1 $TSPPD$ inequalities

Inequalities (24)-(30) are introduced and identified through the separation algorithms described in [9]. The reader is referred to this work for details.

4.2 LIFO inequalities

To separate the LIFO inequalities (33) we proceed as follows. For each pair $i, j \in P$, we check whether there exists a set S such that $i, n+i, n+j \notin S$, $j \in S$ and there is one unit of flow going from i to S and one unit of flow from S to $n+i$. It means that i is visited before S while $n+i$ is visited after S leading to the following delivery pattern: $i < j < n+i < n+j$. To find this set, we use the separation procedure

for (31) proposed in [9]. For each pair $i, j \in P$, we record the 3-tuple (i, j, S), if we can find such a set. Now, since $M + 1$ requests need to cross each other to violate an inequality, we check whether the corresponding items cross each other for each ordered sequence in $\mathcal{L} = \{i_1, i_2, ..., i_{M+1} \in P, i_j \neq i_k\}$ (so that (33) is violated).

The complexity of this separation procedure clearly depends on the complexity of the algorithm used to find the set S for each pair of nodes $i, j \in P$ and on the number of stacks M (as we need to check tuples of $M+1$ nodes). To find set S, for a given pair of nodes i and j, we have to solve a max-flow problem. To this end, we use the procedure proposed in [23] for which the worst-case performance is $O(n^2\sqrt{m})$ on a graph with n nodes and m arcs. However, in practice, the average performance is $O(n^{1.5})$. As there are n^2 pairs of pickup nodes, we obtain a complexity $O(n^2 f(n))$, where $O(f(n))$ is the complexity of the max-flow algorithm. Checking all tuples can be done in $O(n^{M+1})$. Hence, the total complexity is $O(n^{M+1} + n^2 f(n))$.

4.3 Conflict capacity inequalities

The separation of the conflict capacity inequalities (40) is similar to the LIFO inequalities, with the exception that the intersection of all sets must be determined to calculate the value of $z(S_{i_2}, ..., S_{i_k})$. This additional check increases the complexity of the separation procedure, since the sets contain $O(n)$ nodes. To reduce the computation time, this procedure is only used with $k = 2$ or 3.

The procedure finds, for each pair $(i, j) \in P$, a set S such that $i, n+i, n+j \notin S$ and $j \in S$. Then, it checks all ordered sequences in $\mathcal{L} = \{i_1, i_2, ..., i_k \in P, 2 \leq k \leq M-1, i_j \neq i_k\}$ to find k requests that cross each other. If such a sequence is found, the value $z(S_{i_2}, ..., S_{i_k})$ is calculated by scanning the nodes in the intersection $S_{i_2} \cap ... \cap S_{i_k}$, which requires at most $O(n)$ operations. The final complexity of the procedure is $O(n^{k+1} + n^2 f(n))$.

4.4 Rounded capacity inequalities

We use the heuristic procedure described in [28] to separate inequalities (36). This algorithm iteratively builds set S by first choosing a single node to initialize the set and by adding at each iteration the node which maximizes the following function $f(S)$:

$$\begin{aligned} f(S) = \lambda_1(\max\{q(\pi(S)\backslash S), -q(\sigma(S)\backslash S)\} - Qx(\delta^+(S))) + \\ \lambda_2 Q(\max\{\lceil \frac{q(\pi(S)\backslash S}{Q} \rceil, \lceil \frac{-q(\sigma(S)\backslash S)}{Q} \rceil\} - x(\delta^+(S))) + \\ \lambda_3(\min\{q(\pi(S)\backslash S), -q(\sigma(S)\backslash S)\} - Qx(\delta^+(S))). \end{aligned} \tag{42}$$

The algorithm is stopped when an inequality of type (36) is violated or when it

is unlikely that such a violation will be found, as it is done in [28]. The function $f(S)$ contains the three parameters λ_1, λ_2 and λ_3. The first two are randomly set to a value in the interval $[1, 5]$ and the last one in the interval $[0, 1]$. These values are chosen at each restart of the heuristic. In [28], the heuristic is run several times, using each node as a starting point. However, we observed that it is relatively easy to find sets S that violate the inequality in our problem. To avoid generating the same inequality many times, we only run the heuristic five times and randomly choose the starting node each time. We refer the reader to [28] for details.

5 Branch-and-cut algorithm

Different branching strategies proposed in [8] were tried within our branch-and-cut algorithm and the most effective one was the strong branching. We also implemented local pools of cuts, as suggested in [8] and [24], but we observed that they were not really effective. Instead, we separated all $TSPPD$ inequalities (24)-(30) and rounded capacity inequalities (36) at each node of the branch-and-bound tree and added them locally. If none of the previous cuts were generated, we separated the LIFO inequalities (33) and the conflict capacity inequalities (40) and added them locally. Each time a feasible *PDTSP* tour was found, we called the packing procedure to find a feasible packing. This implementation proved to be the most effective one. Unfortunately, adding cuts locally caused the separation procedures to generate the same cuts several times at different branches of the tree. For example, on the *DTSPMS* instances with 2 stacks and a capacity of 7 for each stack, an average of 110 000 cuts were generated. By adding them globally, only 4 000 cuts were generated on average. However, from a computational point of view, it was much better to add the cuts locally.

The upper bound was provided by the large neighborhood search heuristic in [10], which was run 10 times for a total of 25 000 iterations. We do not report the exact computation times for this heuristic, but these times stand between 3 and 30 seconds, depending on the instance.

6 Computational results

The branch-and-cut algorithm was coded in C++ and used Cplex 12 as the integer programming solver. All tests were performed on a 2.2 Ghz AMD Opteron 275 processor running Linux and the maximum computation time was set to one hour (except for some experiments in Section 6.3.1). In the following, the test instances are first described, then a comparison of the three models proposed in Section 2 is presented. Finally, the results obtained with the best model on *PDTSPMS* and *DTSPMS* instances are reported.

6.1 Instances

To test the branch-and-cut algorithm, new *PDTSPMS* instances were generated based on the *PDTSPL* benchmark instances in [3, 5, 9] with 23, 27, 31, 35, 39 and 43 nodes. We generated two classes of instances. In the first class *C1*, the demand of each pickup is one unit, the number of stacks is a random number between 2 and 4 and the capacity of each stack is a random number between 1 and 3. In the second class *C2*, the demand of each pickup is a random number between 1 and 10, the number of stacks is a random number between 2 and 4 and the capacity is a random number between 10 and 15. Note that the capacity value is tight to get difficult instances and obtain solutions that are different from optimal solutions to the standard *PDTSP*. We have a total of 54 instances in each class (i.e., 9 instances of each size).

6.2 Model comparison

We first report some results aimed at evaluating the effectiveness of the three models proposed in this work. These results are summarized in Table 1. M_1 refers to model (1)-(14) and (24)-(30). M_2 is given by (1)-(6), (13)-(14), (15)-(18) and (24)-(30), while M_3 is made of (1)-(5), (13), (19) and (24)-(30). The first group of three lines refer to the plain models. In the following three groups, the rounded capacity constraints (36), LIFO inequalities (33) and conflict capacity inequalities (40) are integrated, respectively. In the last group of three lines all inequalities are integrated. For both classes of instances we report the number of instances solved over a total of 54 instances (*Solved*), the average final gap on unsolved instances (*Gap*), the average root node gap (*Root*), the average CPU time in seconds (*Time*) and the average number of path inequalities (*Path*) generated through the packing procedure in formulation M_3. Gaps are measured with regard to the optimal solution, when available, otherwise they are measured with regard to the heuristic solution found in [10].

The results show that the LIFO and conflict capacity inequalities (41) improve slightly the gaps when compared with the rounded capacity inequalities (36). Moreover, the rounded capacity inequalities seem to cut off most of the LIFO- and capacity-infeasible paths. Using all inequalities brings the largest gain in terms of number of solved instances and gap. Overall, model M_3 is superior to both M_1 and M_2 when all inequalities are considered. Accordingly, formulation M_3 will be used in the following.

6.3 PDTSPMS results

Tables 2 and 3 report detailed results for formulation M_3 on all instances of both classes. The tables present the instance name (*Instance*), the number of nodes

Model	*Class 1*					*Class 2*				
	Solved	*Gap*	*Root*	*Time*	*Path*	*Solved*	*Gap*	*Root*	*Time*	*Path*
M1	21	16.1%	9.9%	192.5		15	27.3%	11.1%	355.8	
M2	24	14.3%	9.8%	273.0		17	24.4%	11.1%	457.8	
M3	20	17.8%	9.9%	279.0	2503.4	13	32.6%	11.0%	109.3	3337.5
M1 + LIFO (33)	22	15.4%	9.8%	314.8		16	25.6%	11.1%	429.8	
M2 + LIFO (33)	24	14.3%	9.8%	261.2		16	25.8%	11.1%	220.5	
M3 + LIFO (33)	21	15.9%	9.8%	276.0	760.9	16	25.0%	11.0%	440.6	1180.1
M1 + Conflict (41)	24	13.8%	9.6%	312.9		16	24.4%	11.1%	332.1	
M2 + Conflict (41)	24	14.3%	9.8%	270.2		17	24.1%	11.1%	373.7	
M3 + Conflict (41)	24	13.3%	9.6%	466.1	715.4	18	20.9%	11.0%	333.2	727.1
M1 + Rounded (36)	29	5.8%	6.1%	298.0		17	15.0%	7.9%	337.9	
M2 + Rounded (36)	28	7.1%	6.6%	319.3		16	17.4%	8.2%	158.6	
M3 + Rounded (36)	31	5.1%	6.3%	266.7	95.4	22	10.1%	7.8%	501.1	767.1
M1 + All	30	5.5%	5.9%	261.2		19	12.9%	7.8%	436.1	
M2 + All	28	7.3%	6.5%	324.2		18	15.0%	8.0%	355.1	
M3 + All	34	4.2%	6.1%	381.9	8.3	24	8.7%	7.8%	312.2	186.5

Table 1: Model comparison

$(2n + 1)$, the number of stacks (M), the capacity of each stack (Q), the heuristic upper bound (UB), the optimal solution ($Optimal$), the gap between the final lower bound and the heuristic upper bound (Gap), the root node gap ($Root$), the CPU time in seconds ($Time$), the number of cuts of type (33), (35) and (40) ($Cuts$), the number of path inequalities (19) generated through the packing procedure ($Path$) and the number of nodes in the branch-and-bound tree ($Nodes$).

More instances in class *C1* are solved when compared to class *C2*, which means that class *C1* contains easier instances. We also observe that the number of added cuts for both classes of instances is quite large, although path inequalities are seldom violated. A large number of path inequalities is observed only in the few cases where the LIFO and conflict capacity inequalities prove to be inefficient.

The sets of instances *brd14051* and *nrw1379* are very difficult to solve. In fact, these instances are also very difficult to solve in their *PDTSP* version. It seems that when some *PDTSP* instance cannot be solved, the same is true for the corresponding *PDTSPMS* instance. The comparison between the heuristic and optimal solutions show that the heuristic algorithm often finds the optimum or is close to the optimum. There are, however, a few cases where the heuristic solution is far from the optimum.

6.3.1 DTSPMS results

Our branch-and-cut algorithm was also applied to the *DTSPMS* test instances in [26], using a CPU time limit of 3 hours. Table 4 summarizes the results, where each row refers to a set of 20 instances. Here, we show the number of stacks (M), the capacity of each stack (Q), the number of nodes (n), the number of solved instances ($Solved$), the average gap for unsolved instances (Gap), the average gap at the root node ($Root$), the average CPU time in seconds for solved instances ($Time$), the number of cuts of type (33), (35) and (40) ($Cuts$), the number of path inequalities

Instance	$2n+1$	M	Q	*UB*	*Optimal*	*Gap*	*Root*	*Time*	*Cuts*	*Path*	*Nodes*
a280	23	2	2	449	449		3.79%	2.0	192	0	42
	27			488	468		3.46%	6.9	168	0	41
	31			613	542		4.59%	105.1	679	0	320
	35			633	624		5.54%	181.3	1 139	0	631
	39			709	669		5.81%	2202.1	3 434	0	1 982
	43			773	n.a.	10.35%	13.98%	3 600	3 702	0	1 923
att532	23	3	3	4 177	4 177		0.93%	0.2	10	0	2
	27			4 937	4 937		1.31%	0.7	7	0	6
	31			5 151	5 151		2.30%	2.9	24	0	17
	35			5 294	5 294		1.51%	2.8	19	0	10
	39			5 587	5 587		2.35%	13.8	30	0	25
	43			9 266	9 266		3.60%	2034.2	3 110	32	4 020
brd14051	23	2	2	4 396	4396		3.69%	8.7	314	0	84
	27			4 439	4 439		4.46%	30.4	642	3	164
	31			4 809	n.a.	3.41%	8.38%	3 600	31 596	0	4 894
	35			4 945	n.a.	4.65%	6.87%	3 600	20 525	0	3 267
	39			6 704	n.a.	4.72%	8.87%	3 600	9 645	0	1 905
	43			6 923	n.a.	4.40%	4.60%	3 600	7 024	0	1 305
d15112	23	3	2	74 603	74 603		6.32%	2.9	98	2	66
	27			80 690	80 690		6.89%	37.0	454	0	265
	31			89 754	89 754		4.82%	20.6	200	8	122
	35			96 804	96 804		6.51%	3 278.0	8 569	66	5 402
	39			103 609	n.a.	5.39%	8.86%	3 600	5 110	1	2 630
	43			109 048	n.a.	9.68%	12.97%	3 600	3 875	0	1 390
d18512	23	2	3	4 280	4 280		1.03%	1.4	46	0	8
	27			4 301	4 301		1.30%	5.7	103	0	32
	31			4 638	4 638		5.84%	2 574.1	15 090	331	7 901
	35			4 741	n.a.	3.56%	6.68%	3 600	9 299	0	4 278
	39			4 917	n.a.	5.78%	8.22%	3 600	5 298	0	3 037
	43			5 100	n.a.	9.02%	11.19%	3 600	3 321	0	1 923
fnl4461	23	4	1	1 889	1 889		0.50%	0.6	54	0	10
	27			2 088	2 088		0.50%	1.6	81	0	8
	31			2 356	2 356		1.78%	16.8	238	0	73
	35			2 517	2 517		3.80%	102.6	560	0	168
	39			2 933	n.a.	3.02%	11.02%	3 600	8 883	0	2 639
	43			3 561	n.a.	3.55%	19.71%	3 600	7 174	0	1 408
nrw1379	23	3	2	2 690	2690		2.16%	0.9	39	0	6
	27			3 061	n.a.	3.68%	10.74%	3 600	21 519	1	10 552
	31			3 117	n.a.	5.63%	10.67%	3 600	18 099	0	7 165
	35			3 197	n.a.	6.21%	10.07%	3 600	11 748	0	4 310
	39			3 476	n.a.	12.11%	14.97%	3 600	7 324	0	2 253
	43			3 799	n.a.	14.69%	16.89%	3 600	5 533	0	1 466
pr1002	23	2	2	13 718	13 718		1.05%	0.4	45	0	7
	27			15 436	15 436		3.34%	5.1	121	0	37
	31			16 268	16 268		5.01%	146.6	1 686	0	813
	35			17 601	17 601		4.30%	384.2	3 143	0	1 484
	39			18 673	18 673		4.33%	1 761.7	9 266	0	4 414
	43			20 199	n.a.	2.31%	5.88%	3 600	4 388	0	2 456
ts225	23	2	2	22 000	22 000		0.00%	1.5	66	3	20
	27			29 395	29 395		0.26%	2.4	51	0	15
	31			32 541	32 541		2.66%	4.0	63	0	10
	35			36 405	36 405		7.78%	44.4	241	1	73
	39			40 395	n.a.	2.18%	10.55%	3 600	5 309	0	3 156
	43			43 056	n.a.	5.65%	13.31%	3 600	4 980	0	2 299

Table 2: Detailed results for the instances in class *C1*

Instance	$2n+1$	M	Q	*UB*	*Optimal*	*Gap*	*Root*	*Time*	*Cuts*	*Path*	*Nodes*
a280	23	2	12	455	455		4.89%	8.6	252	5	82
	27			479	479		4.62%	15.3	450	3	139
	31			592	553		5.95%	497.5	3105	19	1131
	35			655	635		6.93%	1082.3	2509	4	1106
	39			717	n.a.	6.94%	11.57%	3600	7622	11	3266
	43			793	n.a.	12.68%	15.98%	3600	4199	0	1753
att532	23	2	15	4190	4190		0.50%	0.3	17	0	3
	27			5033	5033		3.31%	4.7	97	0	44
	31			5665	n.a.	5.23%	11.08%	3600	13263	0	7842
	35			5920	n.a.	6.98%	11.60%	3600	10772	0	5551
	39			6184	n.a.	7.68%	11.26%	3600	5083	0	2560
	43			10025	n.a.	8.12%	9.85%	3600	4311	0	1566
brd14051	23	3	11	4386	4386		3.45%	4.0	66	0	31
	27			4459	4458		4.87%	960.4	5276	1820	6540
	31			4795	n.a.	3.99%	10.07%	3600	29334	0	5802
	35			4891	n.a.	8.21%	11.25%	3600	20218	0	3569
	39			6276	n.a.	2.81%	5.25%	3600	10015	0	2180
	43			6322	n.a.	3.37%	4.95%	3600	6494	0	1846
d15112	23	3	10	73872	73872		5.40%	5.1	98	12	78
	27			81657	81657		8.52%	452.7	2935	996	3734
	31			91799	91799		7.34%	1402.2	4781	1673	7027
	35			97040	n.a.	3.22%	9.21%	3600	8631	93	5481
	39			99729	n.a.	3.48%	7.88%	3600	5160	8	3784
	43			105242	n.a.	8.50%	11.03%	3600	3048	0	1706
d18512	23	2	14	4341	4341		2.40%	70.6	1467	186	1085
	27			4572	n.a.	2.35%	5.97%	3600	28051	3162	17907
	31			4893	n.a.	2.82%	6.91%	3600	13879	0	4795
	35			5099	n.a.	4.15%	5.84%	3600	14801	0	3672
	39			5359	n.a.	6.48%	9.56%	3600	8993	0	2535
	43			5768	n.a.	12.00%	12.69%	3600	5936	0	1520
fnl4461	23	3	10	1883	1883		0.66%	0.5	30	1	6
	27			2088	2088		2.19%	14.7	332	33	189
	31			2262	2262		2.99%	36.8	502	26	224
	35			2428	n.a.	3.09%	7.31%	3600	12539	2	6083
	39			2634	n.a.	8.35%	11.11%	3600	7019	7	3603
	43			2773	n.a.	8.79%	10.94%	3600	2522	0	1919
nrw1379	23	4	10	2690	2690		2.34%	1.3	38	0	9
	27			3055	n.a.	2.82%	10.48%	3600	27110	5	13395
	31			3116	n.a.	5.62%	10.61%	3600	13525	18	6614
	35			3197	n.a.	6.01%	10.14%	3600	11452	0	4628
	39			3422	n.a.	9.47%	12.02%	3600	4320	0	1482
	43			3769	n.a.	11.33%	13.08%	3600	5390	0	1603
pr1002	23	3	13	13527	13527		1.55%	2.0	39	18	68
	27			15221	15221		3.92%	14.2	170	60	271
	31			15676	15676		3.19%	30.9	333	35	301
	35			17009	17009		2.79%	75.7	574	52	427
	39			18136	18136		3.31%	504.8	2085	66	1558
	43			19613	19613		4.24%	1322.2	4018	95	2650
ts225	23	2	12	22000	22000		0.00%	0.9	39	0	9
	27			34000	34000		10.60%	985.3	9191	468	5419
	31			37703	n.a.	3.44%	10.57%	3600	15077	802	8884
	35			41703	n.a.	5.63%	15.23%	3600	10796	392	6549
	39			45703	n.a.	11.29%	20.03%	3600	4652	0	2348
	43			49097	n.a.	14.18%	20.21%	3600	3476	0	1759

Table 3: Detailed results for the instances in class *C2*

(19) generated through the packing procedure (*Path*) and the number of nodes generated in the branch-and-bound tree (*Nodes*). The results show that instances with up to 48 nodes can be solved. The number of added inequalities is huge, which is mostly due to the increased computation time and the addition of local cuts. We also observe that the problem is much more difficult to solve with 2 stacks than with 4 stacks. While most of the instances with 32 nodes and 2 stacks could not be solved within 3 hours of computation time, the instances with 4 stacks were solved in a few seconds.

M	*Q*	*n*	*Solved*	*Gap*	*Root*	*Time*	*Cuts*	*Path*	*Nodes*
2	5	20	20		4.86%	4.9	1 438.0	1.3	380.8
2	6	24	20		6.97%	179.1	22 347.8	6.7	5 510.4
2	7	28	16	2.66%	8.57%	1 723.3	223 692.9	4.5	49 677.2
2	8	32	3	5.12%	11.07%	4 424.6	321 952.5	0.9	69 695.1
3	4	24	20		1.93%	3.1	580.0	0.9	99.6
3	5	30	20		3.85%	216.9	20 963.4	10.7	2 550.3
3	6	36	15	1.72%	5.30%	3 385.0	238 683.6	16.3	17 539.2
3	7	42	2	3.40%	7.15%	5 721.7	259 180.1	0.7	13 885.8
4	4	32	20		1.27%	14.0	1 196.1	11.5	179.8
4	5	40	20		2.08%	360.7	22 151.4	26.1	1 668.6
4	6	48	5	1.40%	3.83%	4 282.3	180 588.3	22.1	14 762.8
Total			161						

Table 4: Average results for the DTSPMS instances with 3 hours of CPU time

A comparison with the exact approaches for the *DTSPMS* proposed in [4, 22] is reported in Table 5. We note that a processor of 1.6 GHz is used in [22] while a 2.33GHz Intel Core2 Q8200 processor is used in [4]. For each class of instances, we report the number of instances solved by each procedure and the average CPU time in seconds. We can observe that our branch-and-cut algorithm can solve a larger number of instances.

			Lusby et al.		*Carrabs et al.*		*Our B&C*	
M	*Q*	*n*	*Solved*	*Time*	*Solved*	*Time*	*Solved*	*Time*
2	5	20	20	9.5	20	2.7	20	4.9
2	6	24	19	926.8	20	104.2	20	179.1
2	7	28	5	1 424.4	16	2423.2	16	1 723.3
2	8	32					3	4 424.6
3	4	24	20	4.0			20	3.1
3	5	30	20	492.2			20	216.9
3	6	36	*5*	n.a.			15	3 385.0
3	7	42					2	5 721.7
4	4	32					20	14.0
4	5	40					20	360.7
4	6	48					5	4 282.3

Table 5: Comparison with other approaches for the DTSPMS

In Table 6, our algorithm is compared with the branch-and-cut approach of Petersen et al. [26] on their test instances, using one hour of computation time (note that these are the only results reported in [26]). Petersen et al. [26] use an Intel Pentium 4, 2.8 GHz with 2 GB RAM machine for their computational tests. This alternative branch-and-cut algorithm consists in building two optimal

routes, that is, one for the pickups and one for the deliveries. Then, the packing problem associated with the two routes is solved. An optimal solution is obtained if a feasible packing is found. Otherwise, parts of the routes that are responsible for the infeasibility are forbidden through additional path inequalities (19). For each class of instances, where each class is made of 5 instances, we report the number of instances solved, the average gap for unsolved instances and the average CPU time in seconds for both algorithms. Once again, due to the new types of inequalities that we consider, our algorithm is superior by solving more instances and by exhibiting a smaller gap on the unsolved instances.

			Petersen et al.			*Our B&C*		
M	*Q*	*n*	*Solved*	*Gap*	*Time*	*Solved*	*Gap*	*Time*
2	4	16	5		22.8	5		0.5
2	5	20	5		285.0	5		6.9
2	6	24	1	4.85%	1 680.0	5		336.7
2	7	28	0	5.76%		3	2.34%	714.7
3	4	24	5		25.0	5		2.0
3	5	30	4	2.39%	1 737.8	5		114.2
3	6	36	1	5.79%	1 995.0	3	1.41%	1 235.8
3	7	42	0	9.45%		1	4.69%	2 460.1
4	4	32	5		33.8	5		20.0
4	5	40	4	5.57%	1 876.0	5		226.4
4	6	48	0	8.72%		1	1.49%	1 809.5
4	7	56	0	12.03%		0	3.31%	
Total			30			43		

Table 6: Comparison between our branch-and-cut and the one of Petersen et al. [26]

In Petersen et al. [26], the authors also report *DTSPMS* results on modified *PDTSPL* instances. In these instances, the vehicle must pickup all items, return to the depot and then perform the deliveries. All instances have 3 stacks and the computation time is set to one hour. Results are presented in Tables 7 and 8. For the algorithm of Petersen et al., these tables show the final gap for unsolved instances, based on the best known solutions, and the CPU time in seconds, as reported in [26]. For our algorithm, the tables report the heuristic upper bound (UB), the lower bound (LB), which is the value of the optimal solution when the instance is solved, the final gap for unsolved instances (Gap), the gap at the root node ($Root$) and the CPU time in seconds ($Time$). Here, we were able to solve 61 instances out of 81, as compared with 46 instances for the algorithm of Petersen et al.

7 Conclusion

This paper has described the first exact algorithm for solving the pickup and delivery traveling salesman problem with multiple stacks. This branch-and-cut algorithm uses different sets of valid inequalities. Some of these inequalities have been inherited from previous work on pickup and delivery problems, while other new valid inequalities have been derived. The computational results show that we can find the optimum on instances with up to 43 nodes. The comparison with previously

			Petersen et al.		*Our B&C*				
Instance	Q	n	*Gap*	*Time*	*UB*	*LB*	*Gap*	*Root*	*Time*
a280	19	3		2	585	585		0,51%	0,3
	23	4		51	654	654		0,76%	1,5
	27	5		19	696	696		0,72%	0,9
	31	5		31	792	792		1,01%	2,8
	35	6		2277	945	945		1,06%	12,9
	39	7	0,68%	3600	1024	1024		0,98%	32,3
	43	7	1,09%	3600	1103	1103		6,75%	99,6
	47	8	1,57%	3600	1179	1179		6,36%	539,6
	51	9	2,21%	3600	1219	1212	0,57%	2,38%	3600
att532	19	3		2	5361	5361		0,02%	0,1
	23	4		23	6399	6399		0,11%	0,3
	27	5		102	7261	7261		0,36%	0,5
	31	5		320	7562	7562		1,28%	3,9
	35	6	1,60%	3600	7863	7863		1,93%	39,0
	39	7	2,88%	3600	8208	8208		3,11%	949,1
	43	7	3,24%	3600	12639	12638		3,42%	1087,8
	47	8	3,66%	3600	13006	12920,5	0,66%	3,81%	3600
	51	9	3,11%	3600	16214	16042	1,06%	3,23%	3600
brd14051	19	3		0	7897	7897		0,91%	0,0
	23	4		1	8064	8064		0,00%	0,1
	27	5		41	8079	8079		0,04%	0,8
	31	5		3	8196	8196		0,00%	0,5
	35	6	0,32%	3600	8252	8252		0,36%	589,2
	39	7	0,30%	3600	8419	8419		0,36%	213,2
	43	7	0,50%	3600	8442	8442		0,53%	2150,7
	47	8	0,71%	3600	8560	8527	0,39%	0,74%	3600
	51	9	1,52%	3600	8644	8553,5	1,05%	1,54%	3600
d15112	19	3		28	93597	93597		1,49%	0,3
	23	4		39	100489	100489		1,54%	0,9
	27	5		211	108574	108574		1,96%	4,8
	31	5	2,44%	3600	127814	127806		4,03%	124,5
	35	6	3,65%	3600	131421	131408		4,50%	1683,2
	39	7	4,64%	3600	136488	133683	2,06%	5,25%	3600
	43	7	5,67%	3600	139965	135082,67	3,49%	5,91%	3600
	47	8	5,63%	3600	141404	136275	3,63%	5,88%	3600
	51	9	7,27%	3600	149772	140940,31	5,90%	7,45%	3600
d18512	19	3		1	7951	7951		0,00%	0,1
	23	4		1	8023	8023		0,00%	0,2
	27	5		6	8034	8034		0,00%	0,5
	31	5		19	8098	8098		0,00%	1,2
	35	6	0,33%	3600	8151	8151		0,36%	244,9
	39	7	0,42%	3600	8327	8327		0,44%	1493,6
	43	7	0,67%	3600	8482	8456	0,31%	0,70%	3600
	47	8	0,92%	3600	8555	8499	0,65%	0,95%	3600
	51	9	1,34%	3600	8672	8577,67	1,09%	1,36%	3600

Table 7: Comparison between our branch-and-cut and the one of Petersen et al. [26]

			Petersen et al.		*Our B&C*				
Instance	Q	n	*Gap*	*Time*	*UB*	*LB*	*Gap*	*Root*	*Time*
fnl4461	19	3		1	3387	3387		0,09%	0,2
	23	4		9	3430	3430		0,12%	0,5
	27	5		185	3628	3628		0,41%	17,8
	31	5		192	3796	3796		0,37%	17,3
	35	6	0,42%	3600	3853	3853		0,88%	401,1
	39	7	1,14%	3600	4027	4016	0,27%	1,32%	3600
	43	7	2,15%	3600	4147	4101,5	1,10%	2,39%	3600
	47	8	3,36%	3600	4315	4201,57	2,63%	3,59%	3600
	51	9	3,93%	3600	4427	4280,38	3,31%	4,11%	3600
nrw1379	19	3		3	4572	4572		0,00%	0,2
	23	4		17	4733	4733		0,11%	0,7
	27	5		273	4872	4872		0,29%	12,3
	31	5		1230	4984	4984		0,34%	14,3
	35	6	0,33%	3600	5212	5212		0,54%	34,3
	39	7	1,41%	3600	5320	5284,13	0,67%	1,60%	3600
	43	7	1,97%	3600	5543	5476,67	1,20%	2,13%	3600
	47	8	1,98%	3600	5592	5513,5	1,40%	2,11%	3600
	51	9	3,20%	n,a,	6056	5879,88	2,91%	3,27%	3600
pr1002	19	3		0	21498	21498		0,14%	0,1
	23	4		15	22977	22977		0,38%	0,3
	27	5		184	25087	25087		0,87%	2,8
	31	5		929	25899	25899		1,39%	12,1
	35	6		731	27246	27245		2,22%	10,5
	39	7		1733	28196	28196		1,32%	20,6
	43	7		5	29875	29875		0,87%	1,7
	47	8		133	31463	31463		0,30%	7,9
	51	9		5	32319	32319		0,05%	6,4
ts225	19	3		0	34000	34000		0,00%	0,1
	23	4		443	43000	43000		0,00%	0,6
	27	5		2	48440	48440		0,00%	1,0
	31	5		4	50580	50580		0,05%	1,1
	35	6		2	50881	50881		0,07%	1,7
	39	7		17	51371	51371		0,11%	2,5
	43	7		8	52322	52322		0,21%	3,1
	47	8		6	54460	54460		0,46%	4,6
	51	9		808	62688	62688		1,60%	44,5
Average			2,82%	219,8			1,72%		162,3

Table 8: Comparison between our branch-and-cut and the one of Petersen et al. [26] (continued)

reported algorithms for the double TSP with multiple stacks, which is a special case of our problem, also demonstrates the effectiveness of our algorithm.

Acknowledgements. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged. We also want to thank Dr. Manuel Iori for his helpful comments.

References

[1] Balas E., Fischetti M., Pulleyblank W.R., "The precedence-constrained asymmetric traveling salesman polytope", *Mathematical Programming 68*, 241-265, 1995.

[2] Battarra, M., Erdogan, G., Laporte, G., Vigo, D., "The travelling salesman problem with pickups, deliveries and handling costs", *Transportation Science 44*, 383-399, 2010.

[3] Carrabs F., Cerulli R., Cordeau J.-F., "An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading", *INFOR 45*, 223-238, 2007.

[4] Carrabs F., Cerulli R., Speranza M.G., "A branch-and-bound algorithm for the double TSP with two stacks", Preprint n. 4, Dipartimento di Matematica e Informatica, Universitá degli Studi di Salerno, Italy, to appear in *Networks*, 2010.

[5] Carrabs F., Cordeau J.-F., Laporte G., "Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading", *INFORMS Journal on Computing 19*, 618-632, 2007.

[6] Casazza M., Ceselli A., Nunkesser M., "Efficient algorithms for the double traveling salesman problem with multiple stacks", In: *Cologne-Twente Workshop on Graphs and Combinatorial Optimization VIII*, Paris, France, pp. 7–10, 2009.

[7] Cordeau J.-F, "A branch-and-cut algorithm for the dial-a-ride problem", *Operation Research 54*, 573-586, 2006.

[8] Cordeau J.-F., Dell'Amico M., Iori M., "Branch-and-cut algorithm for the pickup and delivery traveling salesman problem with FIFO loading", *Computers & Operations Research 37*, 970-980, 2010.

[9] Cordeau J.-F., Iori M., Laporte G., Salazar-González J.-J., "A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading", *Networks 55*, 46-59, 2010.

[10] Côté, J.-F., Gendreau M., Potvin J.-Y., "Large neighborhood search for the single vehicle pickup and delivery problem with multiple loading stacks", Technical Report CIRRELT-2009-47, Montreal University, Montreal, Canada, 2009.

[11] Doerner K.F., Fuellerer G., Gronalt M., Hartl R.F., Iori M., "Metaheuristics for the vehicle routing problem with loading constraints", *Networks 49*, 294–307, 2007.

[12] Dumitrescu I., Ropke S., Cordeau J.-F., Laporte G., "The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm", *Mathematical Programming 121*, 269-305, 2010.

[13] Erdogan, G., Cordeau, J.-F., Laporte, G., "The pickup and delivery traveling salesman problem with First-In-First-Out loading", *Computers & Operations Research* 36, 1800-1808, 2009.

[14] Felipe A., Ortuno M.T., Tirado G., "The double traveling salesman problem with multiple stacks: A variable neighborhood search approach", *Computers & Operations Research 36*, 2983-2993, 2009.

[15] Fuellerer G., Doerner K.F., Hartl R.F., Iori M., "Ant colony optimization for the two-dimensional loading vehicle routing problem", *Computers & Operations Research 36*, 655-673, 2009.

[16] Fuellerer G., Doerner K.F., Hartl R.F., Iori M., "Metaheuristics for vehicle routing problems with three-dimensional loading constraints", *European Journal of Operational Research*, to appear.

[17] Gendreau M., Iori M., Laporte G., Martello S., "A tabu search algorithm for a routing and container loading problem", *Transportation Science 40*, 342-350, 2006.

[18] Gendreau M., Iori M., Laporte G., Martello S., "A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints", *Networks 51*, 4-18, 2008.

[19] Grötschel M., Padberg M.W., "Polyhedral theory", in Lawler E., Rinnooy Kan A.H.G., Shmoys D.B. (eds) *The Traveling Salesman Problem*, pp. 251-305, Wiley, New York, NY, 1985.

[20] Iori M., Martello S., "Routing problems with loading constraints", *TOP 18*, 4–27, 2010.

[21] Iori M., Salazar-González J.-J., Vigo D., "An exact approach for the vehicle routing problem with two-dimensional loading constraints", *Transportation Science 41*, 253–264, 2007.

[22] Lusby R., Larsen J., Ehrgott M., Ryan D., "An exact method for the double TSP with multiple stacks", *International Transactions in Operational Research 17*, 637–652, 2010.

[23] Lysgaard J., "Reachability cuts for the vehicle routing problem with time windows", *European Journal of Operational Research 175*, 210–223, 2006.

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

[25] Miller C., Tucker A., Zemlin R, "Integer programming formulation of traveling salesman problems", *Journal of the ACM 7*, 326–329, 1960.

[26] Petersen H.L., Archetti C., Speranza M.G., "Exact solutions to the double travelling salesman problem with multiple stacks", *Networks*, available online at DOI: 10.1002/net.20375, 2010.

[27] Petersen H.L., Madsen O.B.G., "The double traveling salesman problem with multiple stacks - Formulation and heuristic solution approaches", *European Journal of Operational Research 198*, 139–147, 2009.

[28] Ropke S., Cordeau J.-F., "Branch and cut and price for the pickup and delivery problem with time windows", *Transportation Science 43*, 267-286, 2009.

[29] Ropke S., Cordeau J.-F., Laporte G., "Models and branch-and-cut algorithms for pickup and delivery problems with time windows", *Networks 49*, 258-272, 2007.

[30] Ruland K.S., Rodin E.Y., "The pickup and delivery problem: Faces and branch-and-cut algorithm", *Computers & Mathematics with Applications 33*, 1–13, 1997.

[31] Toulouse S., Wolfer Calvo R., "On the complexity of the multiple stack TSP, kSTSP", *Theory and Applications of Models of Computation*, *Lecture Notes in Computer Science 5532*, Springer-Verlag, Berlin, 360–369, 2009.

[32] Tricoire, F., Doerner K., Hartl, R., Iori, M., "Heuristic and exact algorithms for the multi-pile vehicle routing problem", *OR Spectrum*, available online at DOI: 10.1007/s00291-009-0179-2, 2009.