CIRRELT

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

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation

An Additive Branch-and-Bound Algorithm for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading

Francesco Carrabs
Raffaele Cerulli
Jean-François Cordeau

June 2007

CIRRELT-2007-12

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

An Additive Branch-and-Bound Algorithm for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading

Francesco Carrabs[1], Raffaele Cerulli[1], Jean-François Cordeau[2,*]

1. Dipartiment di Matematica ed Informatica, Università di Salerno, 84084 Fisciano (SA), Italy
2. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7 and Canada Research Chair in Logistics and Transportation, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Abstract. This paper introduces an additive branch-and-bound algorithm for a variant of the pickup and delivery traveling salesman problem in which loading and unloading operations have to be performed in a Last-In-First-Out (LIFO) order. Two relaxations are used within the additive approach: the assignment problem and the shortest spanning *r-arborescence* problem. The quality of the lower bounds is further improved by a set of elimination rules applied at each node of the search tree to remove from the problem arcs that cannot belong to feasible solutions because of precedence relationships. The performance of the algorithm and the effectiveness of the elimination rules are assessed on instances from the literature.

Keywords. Traveling salesman problem, pickup and delivery, LIFO loading, rear loading, additive branch-and-bound.

Acknowledgements. This work was partly supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant 227837-04. This support is gratefully acknowledged. We are also thankful to three anonymous referees for their valuable comments.

---

* Corresponding author: jean-francois.cordeau@hec.ca

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2007

1. Introduction

This paper addresses a variant of the *Traveling Salesman Problem with Pickup and Delivery* (TSPPD) called the *TSPPD with LIFO Loading* (TSPPDL). The TSPPD is well known. It consist of determining a minimum length tour traveled by a vehicle to service n requests. Each request is characterized by an origin vertex, the *pickup* location, where goods are loaded, and a destination vertex, the *delivery* location, where goods are unloaded. The vehicle starts from a fixed vertex, the *depot*, and returns to it after all requests have been satisfied. Every other vertex has to be visited exactly once, with the additional constraint that the pickup vertex associated with any given request must be visited before the corresponding delivery vertex. This problem has been studied, among others, by Kalantari et al. [1985], Fischetti and Toth [1989], Savelsbergh [1990], Healy and Moll [1995], Ruland and Rodin [1997], and Renaud et al. [2000, 2002]. For a recent survey, see Cordeau et al. [2007b].

In the TSPPDL, the LIFO (Last-In-First-Out) constraints require that the loading and unloading of freight be performed in a LIFO order, i.e., goods being picked up are placed at the rear of the vehicle while a delivery is possible only if the corresponding goods are currently at the rear.

The TSPPDL has applications in the distribution of goods by vehicles having a unique entry and exit point for freight and in situations where rearranging the load is not allowed. This may be the case for safety or physical reasons (e.g., weight, fragility, dimensions) or simply to reduce service time at customer locations. The problem also arises in the routing of automated guided vehicles that use a stack to move items between locations in a plant or warehouse.

There is only a limited literature on the TSPPDL. Volchenkov [1982] has analyzed a planar layout problem with LIFO constraints. The results were later used by Levitin [1986] and Levitin and Abezgaouz [2003]. The latter paper proposes an exact algorithm for the routing of multiple-load automated guided vehicles. This problem is in fact a TSPPDL with the difference that each pickup customer can be associated with more than one delivery customer, and vice-versa. Ladany and Mehrez [1984] have studied a version of the TSPPDL in which the LIFO constraints are relaxed, and their violations are penalized in the objective function. Computational results were presented for very small instances (typically $n = 5$).

More recently, Pacheco [1997a,b] has adapted to the TSPPDL the TSP Or-opt operator (Or [1976]). This operator relocates chains of one, two or three vertices in different positions in the tour. The total number of possible exchanges is $\theta(n^2)$, but Pacheco's adaptation runs in $\theta(n^3)$ time due to the checks needed to find feasible 3-exchanges for the TSPPDL. The author has presented results on random instances with up to 120 customers. Cassani [2004] has introduced a Variable Neighborhood Descent (VND) heuristic based on four local search operators. Finally, three new operators for the TSPPDL were introduced by Carrabs et al. [2007]. These operators are embedded into a Variable Neighborhood Search (VNS) heuristic together with the four operators proposed by Cassani [2004]. Computational results show that the solutions produced by the VNS heuristic are significantly better than those of the VND, at the expense of an increase in computing times.

The first exact approach for the problem studied in this paper was introduced by Pacheco [1994, 1995] who developed a branch-and-bound algorithm derived from the algorithms of Little et al. [1963] and Kalantari et al. [1985] for the TSP and TSPPD, respectively. Cassani [2004] has later introduced a different branch-and-bound algorithm in which lower bounds are computed by solving the minimum spanning tree problem (MSTP) and assignment problem (AP) relaxations. Another method, based on dynamic programming, was also introduced by Ficarelli [2005]. These last two approaches are able to solve instances with up to 23 vertices in less than 20 minutes of computing time.

Very recently, three integer programming formulations and a branch-and-cut algorithm for the TSPPDL were introduced by Cordeau et al. [2007a]. This approach is based on the TSPPD formulation of Ruland and Rodin [1997] and relies on an exponential number of constraints to impose the LIFO policy. Several families of valid inequalities are also used to strengthen the formulation. Exact separation procedures are used to identify violated subtour elimination constraints, precedence constraints and LIFO constraints, while heuristic separation procedures are used for the other families of inequalities. This algorithm is able to solve most instances with up to 43 vertices and some instances with 51 vertices in less than 60 minutes of computing time.

In this paper, we introduce an additive branch-and-bound approach for the TSPPDL. Additive lower bounds were introduced by Fischetti and Toth [1989] who have applied it to the TSP with precedence constraints (TSPPC). This approach has also been applied successfully to the symmetric TSP by Carpaneto et al. [1989] and to the asymmetric TSP by Fischetti and Toth [1992]. In comparison with the branch-and-bound proposed by Kalantari et al. [1985], Pacheco [1994] and Pacheco [1995], our algorithm proposes a new search tree and a different exploration strategy. Moreover, we adapt and introduce new elimination rules which aim to reduce the number of arcs in the residual graph (the graph induced by the vertices not yet inserted in the tour). These elimination rules are based on the precedence relations that arise between the vertices of the graph during the construction of a tour. The search tree and visiting strategy chosen for our branch-and-bound algorithm increase the number of known precedence relations and consequently improve the effectiveness of elimination rules. Cassani [2004] has used the same search tree in his branch-and-bound algorithm. In his case, however, the best results are obtained by constructing the tour in a bidirectional way, i.e., starting from the depot in forward and backward directions. In addition, lower bounds are computed by solving the AP or MSTP relaxations. Except for one case, this algorithm is limited to solving instances with at most 17 vertices. In our algorithm, we replace the MSTP with the shortest spanning r-arborescence problem (r-SAP) which produces better lower bounds. We also combine, through the additive lower bounding approach, the AP and r-SAP, thus generating tighter lower bounds that allow the solution of larger instances. The resulting algorithm is able to solve some instances with 43 vertices.

The remainder of the paper is organized as follows. Section 2 introduces the definitions and notation that are used throughout the paper. Section 3 then introduces an equation to compute the number of feasible solutions of the TSPPDL, and describes the search tree that is explored by the additive branch-and-bound algorithm. This algorithm is then described in detail in Section 4. This is followed by computational results in Section 5, and by the conclusion in Section 6.

2. Definitions and notation

Let $R = \{1, ..., n\}$ be a set of n transportation requests. A request $x \in R$ is composed of a pickup vertex x^+ and a delivery vertex x^-. Let $P = \{1^+, ..., n^+\}$ be the set of pickup vertices and $D = \{1^-, ..., n^-\}$ the set of delivery vertices. We restrict ourselves to the case with a single depot denoted by 0 and we assume that the depot and the pickup and delivery vertices are all different, i.e. $P \cap D = \emptyset$, $0 \notin P$ and $0 \notin D$. Under these assumptions we have that $|P| = |D| = n$. The TSPPDL is defined on a weighted complete digraph $G = (N, A, c)$, where $N = P \cup D \cup \{0\}$ is the vertex set, A is the arc set with $m = |A|$, and c is the cost function defined on A. The cost of arc (x, y) is denoted by $c(x, y)$. The problem is to determine a minimum cost Hamiltonian cycle (or *tour*) T^* on G, subject to the constraint that each pickup vertex i^+ is visited before the associated delivery vertex i^-

and the pickup and delivery operations are executed in a LIFO fashion. A tour is a sequence $(S_0, ..., S_i, ..., S_{2n})$, where S_i denotes the i-th stop of the tour and is defined as follows:

$$S_i = \begin{cases} 0 & \text{if } i = 0 \\ x^+ & \text{if the vehicle picks up request } x \text{ at stop } i \\ x^- & \text{if the vehicle delivers request } x \text{ at stop } i. \end{cases}$$

Given a vertex $a \in N$, we denote by $FS(a)$ and $BS(a)$ the outgoing and ingoing arc sets of vertex a in G. Let T be a tour. Define $pos(a)$ as the position of a in T and define $p(S_i, S_j)$ as the path from S_i to S_j in T. Given a request $x \in R$, the two vertices composing this request are denoted by the couple $[x^+, x^-]$. Two couples $[x^+, x^-]$ and $[y^+, y^-]$ in T are *compatible* if one of the following four *compatibility conditions* is satisfied:

$$pos(x^+) < pos(y^+) < pos(y^-) < pos(x^-) \tag{1}$$

$$pos(y^+) < pos(x^+) < pos(x^-) < pos(y^-) \tag{2}$$

$$pos(x^+) < pos(x^-) < pos(y^+) < pos(y^-) \tag{3}$$

$$pos(y^+) < pos(y^-) < pos(x^+) < pos(x^-). \tag{4}$$

When two couples $[x^+, x^-]$ and $[y^+, y^-]$ are not compatible, we say that there is a *cross* $crs(x, y)$ in the tour (see Figure 1a). Note that the presence of a cross implies that the LIFO constraints are not respected.

We also introduce the following definition that will be used in the description of our algorithm.

Definition 1 *A path $p(S_i, S_j)$ of G is consistent if i) $S_i = 0$ and ii) there exists a feasible tour T of G such that $p(S_i, S_j) \in T$* (see Figure 1b).

Observe that precedence and LIFO constraints are satisfied by any consistent path even though such a path may contain a pickup vertex x^+ without the corresponding delivery vertex x^-. Any feasible tour $T = (S_0, \ldots, S_{2n})$ is a consistent path with $2n + 1$ vertices and an arc from $2n$ to 0.

For any path p on G, let $N(p) \subseteq N$ be the set of vertices visited by p. Given a consistent path $p(0, w)$ of G, we define the *residual graph* $G_w = (N_w, A_w)$, where $N_w = (N \setminus N(p(0, w))) \cup \{w\}$ and $A_w = \{(x, y) : x, y \in N_w\}$. The residual graph G_w is thus the subgraph of G induced by w and the vertices that do not belong to $p(0, w)$.

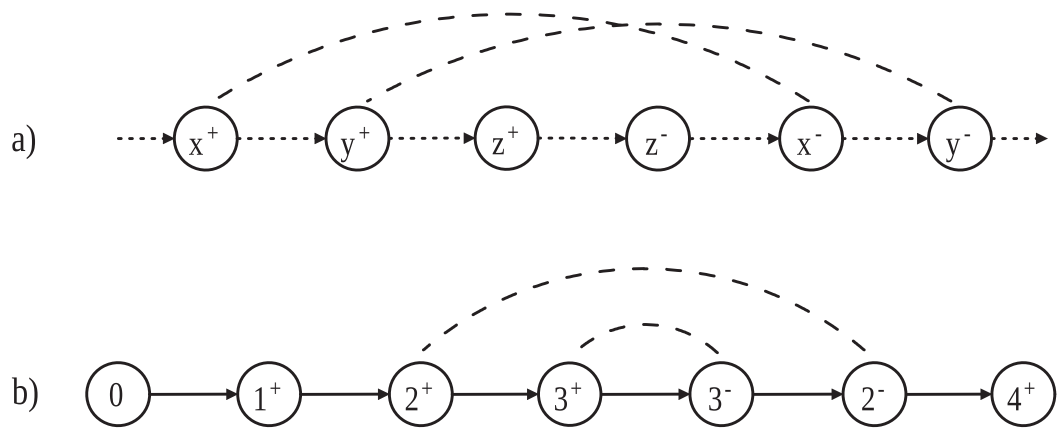

Figure 1: *a) The cross crs(x,y). b) A consistent path p. This path contains pickup vertices* 1^+ *and* 4^+*, but not the corresponding delivery vertices* 1^- *and* 4^-*.*

3. The number of feasible tours

In this section we give an equation to compute, given a directed graph $G = (N, A, c)$, the number of feasible TSPPDL tours on G. We explain in detail how we have derived this equation because the approach followed also underlies the construction of the search tree used in our branch-and-bound algorithm.

Our aim is to construct a tree $\mathcal{T}$ in which each node τ will represent a consistent path on G. Clearly, according to this definition, the number of paths composed by $2n+1$ nodes in $\mathcal{T}$ represents the number of feasible solutions of the TSPPDL on G. The correspondence between a node $\tau \in \mathcal{T}$ and a consistent path in G is defined by the function $\rho(\tau)$ which returns the consistent path of G associated with node τ. We define another function ℓ that, given in input a node τ, returns the last vertex of $\rho(\tau)$. To avoid confusion, we use the term vertex to designate one of the $2n+1$ vertices of G while we will use the term node to denote one of the elements of $\mathcal{T}$.

The root node of the tree $\mathcal{T}$ corresponds to the trivial path containing only the depot vertex (see Figure 2). Because of precedence constraints, the second vertex of any consistent path has to be a pickup vertex. This implies that level 1 of $\mathcal{T}$ contains n nodes: one for each of the n pickup vertices $\{1^+, \ldots, n^+\}$ of G. After selecting a node $\varphi \in \mathcal{T}$ on level 1 with $\ell(\varphi) = x^+$, the consistent path $\rho(\varphi) = [0, x^+]$ of G can be extended in two ways to generate a new consistent path:

- adding the delivery vertex x^- to obtain the path $[0, x^+, x^-]$;

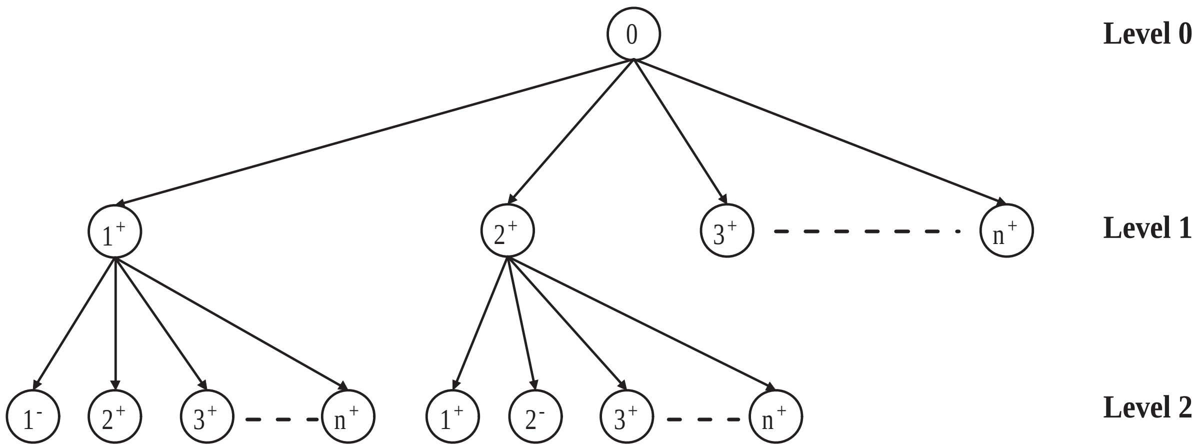

Figure 2: *The tree $\mathcal{T}$ of feasible tours. Level 0 has a single node corresponding to the trivial path containing the depot. Level 1 has n nodes associated with the n pickup vertices of G. At level 2 we show the results of some branching executed on level 1.*

- adding one of the remaining $n-1$ pickup vertices to obtain the path $[0, x^+, y^+]$, where $y^+ \in P \setminus \{x^+\}$.

From these two possibilities, we know that $\rho(\varphi)$ can be extended in n different ways, producing n different nodes at level 2 of $\mathcal{T}$. Since this reasoning holds for each node at level 1 of $\mathcal{T}$, the number of nodes at level 2 is equal to n^2.

In general, given a node $\tau \in \mathcal{T}$ with $\tau \neq 0$, let a^+ be the last pickup vertex visited by $\rho(\tau)$ such that $a^- \notin N(\rho(\tau))$. The branching on the node τ produces a set C_τ of nodes with the following properties: i) $\exists \varphi \in C_\tau$ such that $\ell(\varphi) = a^-$; ii) if $\Gamma = \{x^+ \in P : x^+ \notin N(\rho(\tau))\}$ then $\forall x^+ \in \Gamma\ \exists \psi \in C_\tau$ such that $\ell(\psi) = x^+$; iii) $|\Gamma \cup a^-| = |C_\tau|$.

After completing the construction of $\mathcal{T}$ according to these rules, we can state the following result.

Theorem 1 *The number of nodes on level k of $\mathcal{T}$ is equal to the number of consistent paths of G composed by k vertices plus the depot.*

Proof. The proof is by induction on the level k of $\mathcal{T}$. The base case is for $k = 0$. On level 0 of $\mathcal{T}$ we have only one node and this is correct because there is an unique consistent path composed by zero vertices plus the depot. Assuming that the statement is true for level $k-1$ we want to show that it is also true for level k. In particular we want to prove that to each consistent path of G composed by k vertices plus the depot corresponds a node of $\mathcal{T}$ on level k, and vice-versa.

Let $p = p' \cdot \{a\}$ be a consistent path of G composed by k vertices plus the depot. Since, by the induction hypothesis, there are on level $k-1$ all the consistent paths composed of $k-1$ vertices plus the depot, then there is also a node τ corresponding to the consistent path p'. By construction, the branching on τ generates, on level k of $\mathcal{T}$, the set C_τ of all and only nodes φ such that $\rho(\tau) \cdot \ell(\varphi)$ is a consistent path of G with k vertices plus the depot. This implies that $\exists \varphi \in C_\tau$ such that $\rho(\varphi) = p$.

Conversely, given a node φ on level k of $\mathcal{T}$, the consistent path associated with this node is $\rho(\varphi)$, i.e. the sequence of vertices $\ell(\tau)$ for each node τ belonging to the path from the root of $\mathcal{T}$ to φ. Since this path contains $k+1$ nodes, then $|N(\rho(\varphi))| = k+1$. $\square$

Corollary 1 *The number of leaf nodes in $\mathcal{T}$, which are all located on level $2n$, is equal to the number of feasible tours on G.*

From Corollary 1 we conclude that it is sufficient to count the number of leaves of $\mathcal{T}$ to determine the number of feasible tours of G. In the following we show how to compute the number of leaves of $\mathcal{T}$.

Let $\mathcal{N}(k,x)$ be the number of consistent paths composed by k vertices (plus the depot) of which x are pickup vertices. According to the construction of $\mathcal{T}$ it is easy to see that $\mathcal{N}(1,1) = n$, $\mathcal{N}(0,0) = 1$, and $\mathcal{N}(k,x) = 0$ if $x > k$ or $x < \lceil k/2 \rceil$. This last condition is derived by observing that the number of pickup vertices in a consistent path must be at least equal to the number of deliveries, hence $x \geq \lceil k/2 \rceil$. These conditions represent the base case of our equation. For the remaining cases, the value of $\mathcal{N}(k,x)$ can be computed using the following recursive equation:

$$\mathcal{N}(k,x) = \mathcal{N}(k-1,x) + [\mathcal{N}(k-1,x-1) \times (n-x+1)]. \tag{5}$$

Equation (5) was derived as follows. In general, to construct a consistent path with k vertices, of which x are pickup vertices, one needs to add a vertex to a consistent path p with $k-1$ vertices in which there are either $x-1$ or x pickup vertices. Here we distinguish the following two cases:

- Each consistent path p composed of $k-1$ vertices and containing x pickup vertices can be extended in only one manner, by adding the unique delivery vertex that satisfies the LIFO constraints. This explains the first term in equation (5).

- Each consistent path p composed of $k-1$ vertices and containing $x-1$ pickup vertices can be extended in $(n-x+1)$ ways, by adding one of the $(n-x+1)$ pickup vertices that are not in p. This explains the second term in equation (5).

Using equation (5) we can compute the number of nodes on level k of $\mathcal{T}$ and then, from Theorem 1, the number of consistent paths on G composed by k vertices (plus the depot). Indeed, the number of nodes on level k is equal to the sum of $\mathcal{N}(k,x)$ for $\lceil k/2 \rceil \leq x \leq \min\{k,n\}$. Formally, let $\mathcal{N}(k)$ be the number of nodes on level k. Then,

$$\mathcal{N}(k) = \sum_{x=\lceil k/2 \rceil}^{\min\{k,n\}} \mathcal{N}(k,x). \tag{6}$$

From Equation (6) and Corollary 1 we derive the following claim.

Claim 1 *Given a graph $G = (N, A, c)$ with $|N| = 2n+1$, the number of feasible solutions of TSPPDL on G is given by:*

$$\mathcal{N}(2n) = \mathcal{N}(2n, n). \tag{7}$$

It is interesting to see how much the LIFO constraint reduces the number of feasible solutions of the TSPPDL compared to the classical TSPPD. Using the same reasoning as above, one can easily construct the tree of consistent paths for the TSPPD. Let p be a consistent path (for the TSPPD) with k vertices of which x are pickup vertices. One can extend this path by adding to it any remaining pickup vertex or any delivery vertex whose corresponding pickup vertex is already in p. This is the difference with respect to the TSPPDL in which, because of the LIFO constraints, we can add only one delivery vertex. The number of delivery vertices that we can add to p is equal to $2x+1-k$. Using this idea, we can compute the number of feasible tours for the TSPPD replacing equation (5) with the following:

$$\mathcal{N}(k,x) = \mathcal{N}(k-1,x) \times (2x+1-k) + [\mathcal{N}(k-1,x-1) \times (n-x+1)]. \tag{8}$$

In Table 1 we report the number of solutions for both problems. From this table one can see that the LIFO constraints significantly reduce the number of feasible solutions with respect to the TSPPD.

$\|N\|$	*TSPPD*	*TSPPDL*
3	1	1
5	6	4
7	90	30
9	2.520	336
11	113.400	5.040
13	7.484.400	95.040
15	681.080.400	2.162.160
17	81.729.648.000	57.657.600

Table 1: *The number of feasible solutions for the TSPPD and TSPPDL*

4. An additive branch-and-bound algorithm

In this section we describe our additive branch-and-bound algorithm for the TSPPDL. The three main aspects of a branch-and-bound algorithm are i) the branching strategy (i.e. the construction of the search tree); ii) the exploration strategy for searching the tree; and iii) the computation of lower bounds at each node of the tree. To accelerate the algorithm we also introduce a powerful set of *elimination rules* or *filters* whose aim is to reduce as much as possible the number of arcs in the residual graph considered at each node of the enumeration tree.

We have already described in Section 3 our branching strategy. In the following sections we describe the exploration strategy, the computation of lower bounds, and the set of filters used in the algorithm.

4.1. The exploration strategy

The exploration strategy specifies, after each node evaluation, the node from which the next branching should be performed. The most common strategies are *breadth-first*, *depth-first*, and *best-first*. The *breadth-first* strategy explores the tree level by level, while the *depth-first* strategy explores the tree by visiting at each step a child node of the last one visited. After reaching a leaf, this strategy backtracks to visit the remaining nodes. Finally, the *best-first* strategy selects at each step the node with the smallest lower bound. This strategy usually leads to the early identification of good feasible solutions, thus allowing more pruning of the search tree.

In our algorithm we use a depth-first strategy which is the most efficient in terms of computing time. Indeed, the best-first strategy requires the update of several data structures when jumping from the current node to the more promising leaf in the search tree. Suppose that the algorithm executes the branching on node τ, generating $\mathcal{C}_\tau$. After this, the algorithm identifies the more promising leaf φ of $\mathcal{T}$ and jumps to it (let us suppose that $\varphi \notin \mathcal{C}_\tau$). At this point, the algorithm has to reconstruct the new current path $\rho(\varphi)$ and to apply on φ all the exclusion rules that will remove arcs from the residual graph according to the new set of precedences generated by $\rho(\varphi)$. Finally, one must update the data structures used to represent the residual graph taking into account the arcs removed by the exclusion rules and the vertices outside the current path (except $\ell(\varphi)$). These operations decrease the performance of the algorithm because they are repeated millions of times. The depth-first strategy is much cheaper from a computational point of view because the new current path is obtained by simply extending the old one with one of the vertices in C_τ. The exclusion rules remove arcs from the residual graph by only taking into account the precedences just generated between the last vertex introduced in the current path and the ones already in this path. This reasoning also holds when pruning a node of the search tree.

A good property of the best-first strategy, compared to the depth-first strategy, is that it quickly finds good upper bounds which can reduce the total number of nodes that must be explored in the search tree. However, because we use as an upper bound the solution produced by the VNS heuristic of Carrabs et al. [2007], which often finds the optimal solution on instances with less than 50 vertices, we are able to use a cheaper visiting strategy without increasing the number of nodes in the search tree.

4.2. Lower bound computation

After choosing branching and exploration strategies the final step for the creation of a branch-and-bound algorithm consists in the computation of lower bounds at each node of search tree. This step is essential to prune the search tree and speed up the algorithm. In the following we explain why the lower bounds are so important and how we compute them.

Given a node $\tau \in \mathcal{T}$ with $\tau \neq 0$, let T^* be the best tour found so far and let G_τ be the residual graph induced by $\ell(\tau)$ and the vertices in $N \setminus N(\rho(\tau))$. In order to generate a feasible tour we have to find in G_τ a path p_τ from $\ell(\tau)$ to 0 containing all vertices of G_τ and such that $\rho(\tau) \cdot p_\tau$ is a feasible tour. We call p_τ a *residual path* of G_τ. Let us suppose now that we know a lower bound, lb_τ, on the cost of all residual paths of G_τ. If $c(\rho(\tau)) + lb_\tau \geq c(T^*)$ then each tour with prefix $\rho(\tau)$ will have a cost greater than or equal to the best one found so far and it is therefore useless to continue the construction of these tours as they cannot be better than T^*. This condition allow us to prune the search tree at node τ, avoiding the exploration of the subtree of $\mathcal{T}$ rooted in τ. Obviously, the larger the number of prunings performed on $\mathcal{T}$, the smaller the number of nodes to be explored. For this reason, it is essential to compute lower bounds that are as tight as possible.

For the TSP various relaxations allow the computation of a lower bound on the optimal tour. For instance, two common relaxations used for the TSP are the *1-tree problem* and the *assignment problem*. However, Kalantari et al. [1985] reported that the solutions generated by these two relaxations do not provide tight lower bounds for the TSPPD. The authors adapted the assignment problem to handle the pickup and delivery precedence constraints, but in a preliminary study this approach was found to be ineffective because of the amount of branching required. Clearly, if the lower bounds computed through these relaxations are weak for the TSPPD, they will be even weaker for the TSPPDL. For this reason, we have decided to apply the additive approach, introduced by Fischetti and Toth [1989], and which can be outlined as follows in the context of the TSP.

Let $\mathscr{L}^{(1)}, \mathscr{L}^{(2)}, \ldots, \mathscr{L}^{(q)}$ be q bounding procedures available for the TSP. Suppose that for $h = 1, 2, \ldots, q$ and for any cost matrix $c = (c_{ij})$, procedure $\mathscr{L}^{(h)}(c)$ applied to an instance with cost matrix c returns a lower bound $\delta^{(h)}$ as well as a *residual cost* matrix $c^{(h)} = (c_{ij}^{(h)})$ such that:

i) $c_{ij}^{(h)} \geq 0$ for all $i, j \in N$;

ii) $\delta^{(h)} + \sum_{i \in N} \sum_{j \in N} c_{ij}^{(h)} x_{ij} \leq \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$ for any feasible solution (x_{ij}).

The additive approach generates a sequence of problems, each obtained by considering the residual cost matrix corresponding to the previous problem and applying a different bounding procedure. The procedure is summarized in Figure 3.

An inductive argument shows that the δ values computed in Step 7 of the procedure provide a nondecreasing sequence of valid lower bounds.

Fischetti and Toth [1989] have applied the additive approach to the TSPPC using as bounding procedures the AP and r-SAP. Carpaneto et al. [1989] have applied this approach to the symmetric TSP while Fischetti and Toth [1992] have applied it to the asymmetric TSP. For the TSPPDL we use the same bounding procedures as Fischetti and Toth [1989], i.e., the AP and r-SAP. These two relaxations are derived from the following mathematical

Procedure: *Additive*

input: cost matrix c
output: lower bound δ and the corresponding residual-cost matrix $c^{(q)}$

$c^{(0)} \leftarrow c; \quad \delta \leftarrow 0;$
for $h = 1$ to q **do**
  apply $\mathscr{L}^{(h)(c^{(h-1)})}$ obtaining $\delta^{(h)}$ and the residual cost matrix $c^{(h)}$
  $\delta \leftarrow \delta + \delta^{(h)};$
end for

Figure 3: Additive approach

formulation of the ATSP (see, e.g., Gutin and Punnen [2002]):

$$\text{Min} \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{9}$$

subject to

$$\sum_{i \in N} x_{ij} = 1 \qquad \forall j \in N \tag{10}$$

$$\sum_{j \in N} x_{ij} = 1 \qquad \forall i \in N \tag{11}$$

$$\sum_{i \in S} \sum_{j \in N \setminus S} x_{ij} \geq 1 \qquad S \subset N : S \neq \emptyset \tag{12}$$

$$x_{ij} \geq 0 \qquad \forall i, j \in N \tag{13}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in N, \tag{14}$$

where $x_{ij} = 1$ if and only if arc (i, j) is in the tour. Constraints (10) and (11) restrict the in-degree and out-degree of each vertex to be equal to one, while constraints (12) impose strong connectivity. It is well known that one can halve the number of constraints (12) by replacing them with:

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \qquad S \subset N : r \in S. \tag{15}$$

Constraints (10), (11) and (13) with objective function (9), define the AP. This problem always has an integer optimal solution and requires finding a minimum-cost collection of vertex-disjoint *subtours* visiting all the vertices of G. If an optimal solution of AP determines only one directed cycle, then it satisfies all constraints (12) and is thus optimal for the ATSP as well. Relaxation AP can be solved in $O(n^3)$ using the Hungarian algorithm (see, e.g., Ahuja et al. [1993]).

Constraints (10), (13) and (15) with the objective function (9), define the r-SAP problem. Formally, the r-SAP is defined as follows. Given a graph $G = (N, A)$ and a root vertex r, the shortest spanning r-arborescence problem consists in finding a minimum cost spanning sub-graph $G' = (N, A')$ of G such that: i) the in-degree of each vertex is exactly one, and ii) each vertex can be reached from the root r. If an optimal solution of r-SAP leaves each vertex with out-degree equal to 1, then it satisfies all constraints (11) and is thus optimal for the ATSP as well. Relaxation r-SAP can be solved in $O(n^2)$ time by finding the shortest spanning arborescence rooted at vertex r (SSA_r) and by adding to it a minimum-cost arc entering vertex r. Since, unlike the MSTP, this problem is solved on a directed graph the lower bound produced in this way may be tighter. Polynomial algorithms for solving SSA_r have been proposed, independently, by Chu and Liu [1965] and by Edmonds [1967].

Tarjan [1977] gave efficient implementations of Edmonds' algorithm, requiring $O(|N|^2)$ time for complete digraphs, and $O(|A| \log |N|)$ time for sparse digraphs. Camerini et al. [1979] have corrected an error in Tarjan's implementation. Different implementations for sparse digraphs based on sophisticated data structures have been proposed by Gabow et al. [1986, 1989]. For our implementation we followed the guidelines presented by Gabow et al. [1986] and by Fischetti and Toth [1993] to accelerate the first phase (contraction phase) of Edmonds' algorithm.

To obtain tighter lower bounds, an enhanced relaxation of r-SAP can be introduced. This new relaxation is obtained from r-SAP by adding constraint (11) for $i = r$, i.e., $\sum_{j \in N} x_{rj} = 1$, which imposes an out-degree equal to 1 for the root vertex r. This constraint can be easily introduced by adding a large positive value M to the costs $c(r, j)$, $\forall j \in N$. If v is the value of an optimal solution for the r-SAP with the new cost matrix and t is the number of arcs outgoing from r in this solution then $v - Mt$ is the optimal value for r-SAP with the original cost matrix.

4.3. Filters

The AP and r-SAP used in the additive approach are relaxations of the ATSP, which is itself a relaxation of the TSPPDL. For this reason the lower bounds provided by the additive approach may not be so tight. In the hope of improving the quality of the lower bounds, we introduce a set of filters to detect and remove as many arcs as possible from the residual

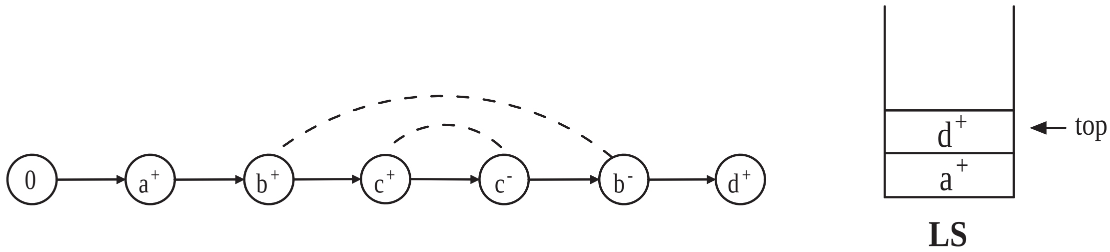

Figure 4: *On the left: the consistent path composed by vertices a^+, b^+, c^+, c^-, b^-, d^+. On the right: the corrensponding LIFO stack LS.*

graph $G_\tau = (N_\tau, A_\tau)$ considered at node $\tau \in \mathcal{T}$. Because of the precedence relations among the vertices in the current path $\rho(\tau)$ and the precedence and LIFO constraints, some arcs cannot belong to a feasible tour with prefix $\rho(\tau)$ and can thus be removed from the graph. Since the "filtered" residual graph contains fewer arcs than G_τ, the solution of our two relaxations on this new graph produces a lower bound that should be closer to the value of shortest residual path p_τ.

Before listing our eight filters, we introduce some further definitions. Let a and b be two vertices of $N(\rho(\tau))$. If $pos(a) < pos(b)$ we say that a *precedes* b in the tour and we denote this by $a \rightsquigarrow b$. Define a stack $LS = \{a^+ : a^+ \in N(\rho(\tau)) \text{ and } a^- \notin N(\rho(\tau))\}$ such that the insertion order of pickup vertices in this stack coincides with their insertion order in $\rho(\tau)$. Consequently, the vertex at the top of LS is the last pickup vertex whose delivery is not in the current path (see Figure 4).

The eight filters applied in our branch-and-bound algorithm are:

$f1$. (Basic) Remove the arcs (a^+, b^-) with $a \neq b$, (a^-, a^+), $(0, b^-)$, $(a^+, 0)$ and (a, a) $\forall a^+ \in P$ and $b^- \in D$. This filter removes from G all arcs that cannot belong to any feasible tour for the TSPPDL. This filter can be directly applied to the graph prior to the construction of the search tree.

$f2$. If $a^+ \rightsquigarrow b^+$ in $\rho(\tau)$ or if $a^+ \in N(\rho(\tau))$ and $b^+ \notin N(\rho(\tau))$, then remove (a^-, b^-).

If $a^+ \rightsquigarrow b^+$ we have two cases to consider: $a^- \rightsquigarrow b^+$ and $b^+ \rightsquigarrow a^-$. In the first case we can trivially remove (a^-, b^-) because, from the precedence constraints, $b^+ \rightsquigarrow b^-$. In the second case, both a^+ and b^+ are in LS and in particular b^+ is over a^+ in the stack (Figure 5a). This implies that b^- has to precede a^- to satisfy the LIFO constraint and then the arc (a^-, b^-) cannot be in the tour we are constructing. Similar reasoning holds when $a^+ \in N(\rho(\tau))$ and $b^+ \notin N(\rho(\tau))$.

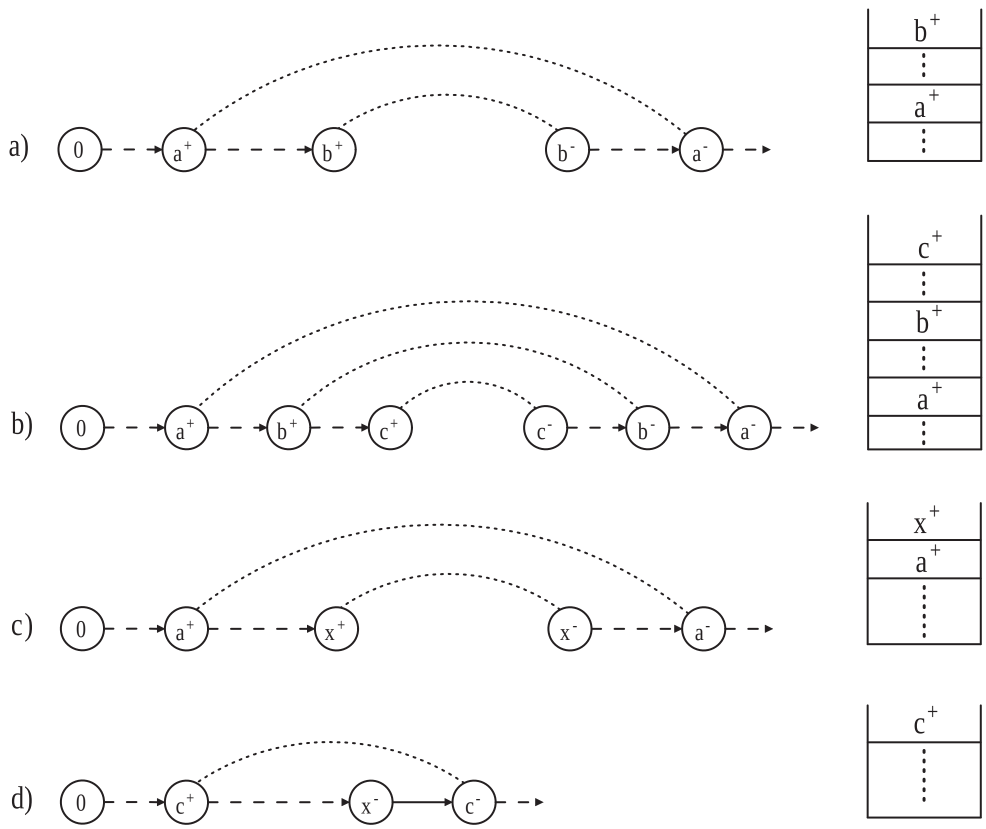

Figure 5: *Graphical representation of some cases of filters f2, f3, f5 and f6.*

$f3$. If $a^+ \rightsquigarrow b^+ \rightsquigarrow c^+$ and $a^+, b^+, c^+ \in LS$ then remove (c^-, a^-).

Under these conditions, to satisfy the precedence and LIFO constraints, we must have that $c^- \rightsquigarrow b^- \rightsquigarrow a^-$ in the final tour (Figure 5b). This implies that the arc (c^-, a^-) cannot be in the tour.

$f4$. If $a^+ \rightsquigarrow b^+$ and $a^+, b^+ \in LS$ then remove $(b^-, 0)$.

Using the arc $(b^-, 0)$ under the previous conditions means completing the tour without inserting in it the vertex a^-. Consequently, the tour produced is not feasible.

$f5$. If $\ell(\tau) = x^+$, for any $x^+ \in P$, and a^+ is the vertex immediately under x^+ in LS, then remove (x^-, c^-) $\forall c^- \in N_\tau \setminus \{a^-\}$.

Because of the position of a^+ and x^+ in LS only the delivery vertex a^- can be added to the current path immediately after x^- to avoid the generation of crosses (Figure 5c). For this reason it is possible to remove all the arcs outgoing from x^- toward the delivery vertices of the residual graph, except a^-.

$f6$. If $\ell(\tau) \in D$ then remove $(\ell(\tau), b^-)$ $\forall b^- \in N_\tau \setminus \{c^-\}$ where c^+ is the vertex at the top of LS.

Since the vertex c^+ is at the top of LS, the only delivery vertex that can be inserted immediately after x^- is c^-. Any other delivery vertex of the residual graph produces a cross with request c (Figure 5d). For this reason, we can remove all the arcs outgoing from x^- toward the delivery vertices of the residual graph, except c^-.

$f7$. Remove $(a, \ell(\tau))$ $\forall a \in N_\tau$.

Since each vertex in a tour has only one ingoing arc and $\ell(\tau)$ already has one because it is inserted in the current path, we can remove all the arcs coming from the residual graph and ingoing to $\ell(\tau)$.

$f8$. If $\ell(\tau) \in D$ and $|N \cap N(\rho(\tau))| \neq 2n + 1$, then remove $(\ell(\tau), 0)$.

If there are other vertices to introduce in the current path to construct a feasible tour, we can clearly remove the arc $(\ell(\tau), 0)$ that produces an infeasible tour.

4.4. The algorithm

Let $G = (N, A, c)$ be a directed and weighted graph and let T^* be a feasible tour identified by a heuristic. The branch-and-bound algorithm starts the generation of the search tree $\mathcal{T}$ from the depot vertex. Given a generic node $\tau \in \mathcal{T}$ with $\tau \neq 0$, the algorithm branches according to the rules described in Section 3, generating the node set $\mathcal{C}_\tau$ (Figure 6).

After generating $\mathcal{C}_\tau$ the algorithm randomly selects a new node $\varphi \in \mathcal{C}_\tau$ from which to perform the next branching. Before executing this branching, the algorithm computes the lower bound lb_φ to verify whether there may exist in the subtree rooted in φ at least one solution better than T^*. To this end, the algorithm applies on the residual graph $G_\varphi = (N_\varphi, A_\varphi)$ the filters described in Section 4.3, removing a set of arcs $H \subset A_\varphi$. At this point the algorithm solves two problems on the filtered residual graph $\widehat{G_\varphi} = (N_\varphi, \widehat{A_\varphi})$, where $\widehat{A_\varphi} = A_\varphi \setminus H$: first the AP relaxation, yielding a temporary lower bound δ' and the residual cost matrix $\overline{c}$; then the r-SAP relaxation on $\overline{c}$, yielding another lower bound δ'' which is added to δ' to produce the lower bound δ on $\widehat{G_\varphi}$. The lower bound on the node φ of the search tree is given by $lb_\varphi = c(\rho(\varphi)) + \delta$. Notice that since the residual path p_φ starts from φ and ends at the depot, before solving our relaxation on $\widehat{G_\varphi}$ we replace $BS(\varphi)$ with the set of arcs $\{(x, 0) : x \in N_\varphi\}$.

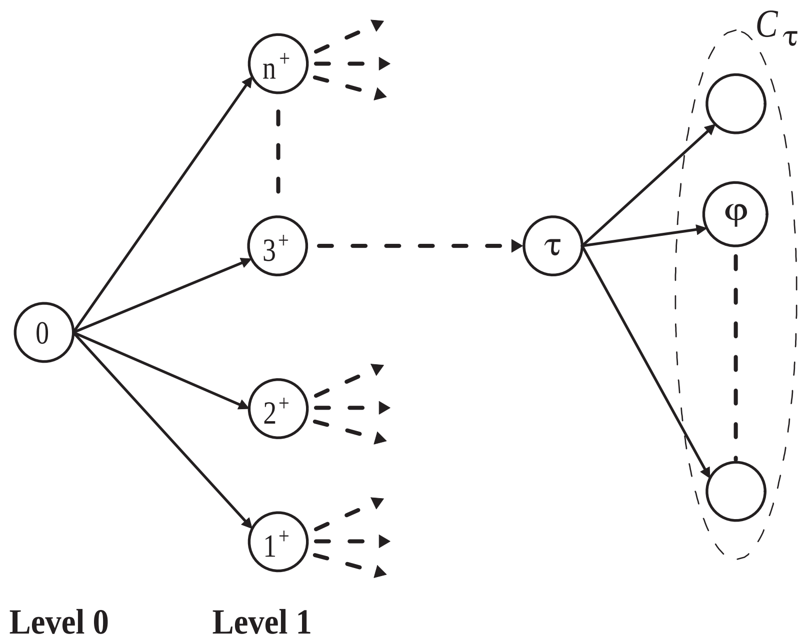

Figure 6: *The search tree $\mathcal{T}$. On the right the branching of τ that produces the children set $\mathcal{C}_\tau$.*

After computing lb_φ, the algorithm checks whether $lb_\varphi \geq c(T^*)$. If this is the case the algorithm prunes node φ and selects a new node in $\mathcal{C}_\tau$ from which to restart the branching. Otherwise, the algorithm branches on φ, generating the new children set $\mathcal{C}_\varphi$ and selecting from it a new node for the next branching. When the current path $\rho(\tau)$ is composed by $2n+1$ vertices, the algorithm completes the tour by adding to it the arc $(\ell(\tau), 0)$ and generates a feasible tour T' for TSPPDL. If $c(T') < c(T^*)$ then the algorithm sets $T^* = T'$.

5. Computational results

Our additive branch-and-bound algorithm was coded in C and run on a 2.4 Ghz AMD Opteron 250 processor. Following the approach used by Carrabs et al. [2007], we have generated test instances for the TSPPDL by adapting instances from TSPLIB. To this end, nine files were used: *a280, att532, brd14051, d15112, d18512, fnl4461, nrw1379, pr1002, ts225*. In each case, seven subsets of customers were selected to yield instances containing *19, 23, 27, 31, 35, 39* and *43* vertices, respectively. For an instance with n vertices, the cost matrix was obtained by considering the first n rows associated with cities in the file. For each file, the pairing of pickup and delivery vertices for the smallest instance ($n = 19$) was obtained by performing a random matching between the selected locations. Larger instances

Instance	**Size**		**BBL**		*Arborescence*		*Assignment*		*GAP*	
name	$\lvert N \rvert$	**UB**	Visited nodes	Time	Visited nodes	Time	Visited nodes	Time	$Ad \rightarrow Ar$	$Ad \rightarrow As$
a280	19	402	2.985	0,02	32.457	0,17	8.469	0,03	88,24	33,33
	23	468	8.915	0,13	143.733	1,11	39.179	0,21	88,29	38,10
	27	505	42.789	1,00	1.180.665	12,26	119.632	0,96	91,84	-4,00
	31	560	126.018	4,21	4.121.411	58,54	570.088	6,09	92,81	30,87
	35	647	545.528	24,42	50.828.395	978,96	3.214.238	45,29	97,51	46,08
	39	691	5.329.064	299,11	375.507.586	8443,55	34.267.257	583,75	96,46	48,76
	43	752	55.792.327	3918,54	n.d.	n.d.	463.362.024	9471,25		58,63
att532	19	4250	4.263	0,03	28.135	0,13	59.675	0,18	76,92	83,33
	23	5038	40.643	0,46	646.965	4,02	624.408	2,83	88,56	83,75
	27	5800	388.248	6,95	11.551.146	105,09	12.689.262	78,96	93,39	91,20
	31	6173	3.352.943	83,33	188.051.653	2358,86	162.217.084	1388,40	96,47	94,00
	35	6361	54.144.429	1671,03	n.d.	n.d.	n.d.	n.d.		
	39	6725	206.598.551	8478,59	n.d.	n.d.	n.d.	n.d.		
	43	10714	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
brd14051	19	4555	247.539	1,22	917.963	2,46	1.246.916	2,69	50,41	54,65
	23	4655	806.696	8,56	8.097.120	30,16	2.256.461	11,52	71,62	25,69
	27	4936	10.925.699	149,00	135.505.592	796,55	69.079.536	446,69	81,29	66,64
	31	5186	14.874.093	367,95	n.d.	n.d.	321.342.607	3062,15		87,98
	35	5196	149.453.979	4958,67	n.d.	n.d.	n.d.	n.d.		
	39	5629	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
	43	5719	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
d15112	19	76203	13.068	0,10	221.961	0,85	39.150	0,13	88,24	23,08
	23	88272	200.554	2,20	10.368.678	53,74	822.409	3,94	95,91	44,16
	27	93158	814.850	15,82	155.512.347	1299,02	5.547.541	43,59	98,78	63,71
	31	109166	13.998.866	391,49	415.546.398	5737,71	87.276.910	913,75	93,18	57,16
	35	115554	45.984.827	1935,22	n.d.	n.d.	372.957.673	5793,58		66,60
	39	119863	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
	43	128798	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
d18512	19	4446	7.436	0,04	331.311	0,97	767.535	1,59	95,88	97,48
	23	4658	634.066	6,24	2.512.323	12,28	1.894.524	8,66	49,19	27,94
	27	4704	12.023.434	141,47	19.353.777	146,80	63.179.735	347,11	3,63	59,24
	31	5120	24.459.889	527,21	365.265.141	3528,83	122.335.390	1234,18	85,06	57,28
	35	5186	335.729.113	9252,33	n.d.	n.d.	n.d.	n.d.		
	39	5419	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
	43	5634	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
fnl4461	19	1866	996	0,01	1.235	0,00	5.871	0,02		50,00
	23	2067	9.444	0,15	37.177	0,27	62.298	0,45	44,44	66,67
	27	2483	154.020	3,56	2.985.419	29,88	1.596.609	15,12	88,09	76,46
	31	2672	575.082	18,93	26.900.815	385,81	14.003.558	181,80	95,09	89,59
	35	2852	4.124.716	178,50	421.376.576	8185,39	126.162.492	2037,76	97,82	91,24
	39	3109	120.707.943	6372,41	n.d.	n.d.	n.d.	n.d.		
	43	3269	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
nrw1379	19	2691	5.571	0,05	1.019.856	2,86	13.014	0,04	98,25	-20,00
	23	2919	11.722	0,18	22.885.165	92,37	15.686.374	55,36	99,81	99,67
	27	3366	18.347.920	311,32	1.253.002.900	6765,31	311.695.166	1937,43	95,40	83,93
	31	3554	299.732.802	7232,92	n.d.	n.d.	n.d.	n.d.		
	35	3652	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
	39	4002	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
	43	4282	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.
pr1002	19	12947	1.306	0,01	3.010	0,02	14.925	0,07	50,00	85,71
	23	13872	2.202	0,04	10.838	0,09	69.980	0,64	55,56	93,75
	27	15566	13.882	0,43	129.589	1,83	777.791	9,85	76,50	95,63
	31	16255	117.976	4,59	732.124	15,09	8.388.684	133,33	69,58	96,56
	35	17564	557.678	28,75	8.006.178	225,55	66.545.439	1361,79	87,25	97,89
	39	18862	9.266.715	563,59	110.463.870	4068,64	n.d.	n.d.	86,15	
	43	20173	29.966.621	2435,54	n.d.	n.d.	n.d.	n.d.		
ts225	19	21000	3.698	0,03	9.697	0,04	5.431	0,01	25,00	-66,67
	23	25000	11.146	0,16	62.181	0,43	18.030	0,09	62,79	-43,75
	27	32395	345.338	5,32	1.520.839	15,63	8.323.803	39,01	65,96	86,36
	31	33395	686.365	20,74	841.602	17,05	5.713.115	47,30	-17,79	56,15
	35	36784	3.366.853	149,39	6.406.906	191,76	19.963.941	255,40	22,10	41,51
	39	39395	12.705.633	752,21	27750950	1099,44	93.497.199	1544,21	31,58	51,29
	43	43082	85.718.933	6120,99	240.369.346	10576,00	n.d.	n.d.	42,12	

Table 2: Performance comparison between the additive algorithm, and the algorithms that use either the arborescence or the assignment relaxation.

($n = 23, 27, \dots$) were then obtained sequentially by performing a random matching between the new locations considered in each step.

Table 2 reports results obtained with our complete additive branch-and-bound algorithm (denoted by *BBL* in the table). It also reports corresponding results obtained by considering either the arborescence relaxation alone or the assignment problem relaxation alone. These latter two algorithms are denoted by *Arborescence* and by *Assignment*, respectively. In this table, column *UB* reports the upper bound given as an input to the algorithm. Except for *ts225-d35* for which the optimum value is equal to 36703, all values reported in column UB coincide with the optimal objective function value. For each algorithm we report the number of nodes visited in the search tree and the total CPU time, in seconds, spent to compute the optimal solution. A maximum CPU time of 3 hours was imposed for the solution of each instance. When an instance could not be solved within that time limit, this is indicated by *n.d.* The last column, *GAP*, shows the difference of computing time, in percentage, between *BBL* and *Arborescence* and between *BBL* and *Assignment*, respectively.

A comparison of the number of nodes visited by each of the three algorithms shows the superiority of the additive approach. Except for one case, *fnl4461-d15*, the number of nodes visited by *BBL* is much smaller than with the two other algorithms. This difference is sometimes dramatic. For example on *a280-d39, att532-d31, brd14051-d27, d15112-d31, d18512-d31, fnl4461-d35, nrw1379-d27, pr1002-d35*, i.e., the largest instances for which all three algorithms reach an optimal solution in less than three hours, the reduction is larger than 80%, and on *ts225-d39* it is equal to 54% with respect to *Arborescence* and to 86% with respect to *Assignment*. Obviously, this reduction in the number of nodes visited often implies that *BBL* is much faster than *Arborescence* and *Assignment*. However, this is not always true because of the extra time spent in solving two relaxations at each node of the tree. From Table 2, one can see that *BBL* is slower than *Arborescence* in only one case, on *ts225-d31*. With respect to *Assignment*, *BBL* is slower in four cases, on *a280-d23*, *nrw1379-d19*, *ts225-d19* and *ts225-d23*. In these cases, however, the gap between *BBL* and *Assignment* is negligible.

For remaining instances, comparing running times reveals that significant improvements are obtained by applying the additive approach. On all *a280, att532, brd14051, d15112, nrw1379* and *pr1002* instances, *BBL* is at least 50% faster than *Arborescence* and this improvement often exceeds 80%. Within the maximum time limit, *BBL* can solve instances with four more vertices on *a280, d15112, d18512, fnl4461, nrw1379, pr1002* and eight more vertices on *att532, brd14051*. The *Assignment* has better performance with respect to *Arborescence*. Nevertheless, the results show that there is a large difference in running times between these two algorithms. Indeed, in all cases for which *BBL* is faster than *Assignment* the gap is at least equal to 25%. Moreover, *BBL* solves instances with four more vertices than *Assignment* on *brd14051, d18512, fnl4461, nrw1379, ts225* and eight more vertices on *att532* and *pr1002*.

Another aspect that we have studied is the impact of filters on *BBL*. To this end we have removed from the algorithm all the filters except the basic one (i.e., $f1$). We denote by *BBL_nf* this new algorithm. Table 3 reports the results obtained with *BBL_nf*. These results show the effectiveness of the filters and how much they affect the performance of *BBL*. Except for *pr1002-d23* and *pr1002-d27*, all remaining instances are solved at least 50% faster with *BBL* than with *BBL_nf*. In particular, on the *a280, brd14051, att532, d15112* and *ts225* instances the improvement provided by the filters is at least 70%. Even more relevant is the dimension of instances that can be solved by using the filters. *BBL* solves instances with four more vertices on *brd14051, d18512* and *pr1002*, and with eight more vertices on *a280, att532, d15112, fnl4461, nrw1379* and *ts225*.

Making direct comparisons with existing branch-and-bound algorithms for the TSPPDL is difficult because of the different computers and test instances used. The algorithm of Pacheco [1995] was able to solve instances with at most 17 vertices, on a 486 DX2 50Mhz, in 700 seconds. That proposed by Cassani [2004] was able to solve, on a 500 Mhz Intel PENTIUM 3 Processor, instances with the same dimension in less than 20 seconds. On the fnl4461 instance with at most 23 vertices, this algorithm computed the exact solution in less than 160 seconds. Finally the dynamic programming approach introduced by Ficarelli [2005] solved instances with up to 22 vertices in less than 1150 seconds on a 1.4 GHz Intel Pentium-M 710. Unfortunately, none of these authors reported results for larger instances. With our algorithm, we were able to solve all instances with 23 vertices in less than 10 seconds. With the exception of nrw1379-d31, we were also able to solve all instances with 31 vertices in less than 600 seconds. It thus seems fair to say that our new branch-and-bound algorithm outperforms all previous branch-and-bound approaches and that the improvement in performance cannot be attributed solely to an increase in computing speed.

Finally, Cordeau et al. [2007a] have tested their branch-and-cut algorithm on the same instances used in this paper and they have compared their results with those reported here. The branch-and-cut algorithm could solve most instances with up to 43 vertices within one

Instance	Size	UB	BBL		*BBL_nf*		GAP
			Visited nodes	Time	Visited nodes	Time	
a280	19	402	2.985	0,02	44.475	0,39	94,87
	23	468	8.915	0,13	157.903	2,15	93,95
	27	505	42.789	1,00	1.375.702	27,68	96,39
	31	560	126.018	4,21	6.099.268	173,44	97,57
	35	647	545.528	24,42	41.132.729	1599,31	98,47
	39	691	5.329.064	299,11	n.d.	n.d.	
	43	752	55.792.327	3918,54	n.d.	n.d.	
att532	19	4250	4.263	0,03	103.354	0,76	96,05
	23	5038	40.643	0,46	1.187.710	13,19	96,51
	27	5800	388.248	6,95	13.284.313	242,45	97,13
	31	6173	3.352.943	83,33	289.092.559	6759,83	98,77
	35	6361	54.144.429	1671,03	n.d.	n.d.	
	39	6725	206.598.551	8478,59	n.d.	n.d.	
	43	10714	n.d.	n.d.	n.d.	n.d.	n.d.
brd14051	19	4555	247.539	1,22	1.580.565	8,71	85,99
	23	4655	806.696	8,56	2.484.690	29,61	71,09
	27	4936	10.925.699	149,00	65.132.547	1042,48	85,71
	31	5186	14.874.093	367,95	221.053.635	5656,15	93,49
	35	5196	149.453.979	4958,67	n.d.	n.d.	
	39	5629	n.d.	n.d.	n.d.	n.d.	n.d.
	43	5719	n.d.	n.d.	n.d.	n.d.	n.d.
d15112	19	76203	13.068	0,10	70.979	0,62	83,87
	23	88272	200.554	2,20	3.508.314	41,25	94,67
	27	93158	814.850	15,82	25.521.575	510,44	96,90
	31	109166	13.998.866	391,49	n.d.	n.d.	
	35	115554	45.984.827	1935,22	n.d.	n.d.	
	39	119863	n.d.	n.d.	n.d.	n.d.	n.d.
	43	128798	n.d.	n.d.	n.d.	n.d.	n.d.
d18512	19	4446	7.436	0,04	13.364	0,09	55,56
	23	4658	634.066	6,24	1.366.535	12,37	49,56
	27	4704	12.023.434	141,47	29.251.676	356,69	60,34
	31	5120	24.459.889	527,21	107.338.428	2379,15	77,84
	35	5186	335.729.113	9252,33	n.d.	n.d.	
	39	5419	n.d.	n.d.	n.d.	n.d.	n.d.
	43	5634	n.d.	n.d.	n.d.	n.d.	n.d.
fnl4461	19	1866	996	0,01	3.510	0,03	66,67
	23	2067	9.444	0,15	86.757	1,12	86,61
	27	2483	154.020	3,56	5.761.975	103,17	96,55
	31	2672	575.082	18,93	197.112.308	4822,09	99,61
	35	2852	4.124.716	178,50	n.d.	n.d.	
	39	3109	120.707.943	6372,41	n.d.	n.d.	
	43	3269	n.d.	n.d.	n.d.	n.d.	n.d.
nrw1379	19	2691	5.571	0,05	17.489	0,15	66,67
	23	2919	11.722	0,18	150.690	2,11	91,47
	27	3366	18.347.920	311,32	n.d.	n.d.	
	31	3554	299.732.802	7232,92	n.d.	n.d.	
	35	3652	n.d.	n.d.	n.d.	n.d.	n.d.
	39	4002	n.d.	n.d.	n.d.	n.d.	n.d.
	43	4282	n.d.	n.d.	n.d.	n.d.	n.d.
pr1002	19	12947	1.306	0,01	1.328	0,01	0,00
	23	13872	2.202	0,04	2.962	0,05	20,00
	27	15566	13.882	0,43	14.988	0,45	4,44
	31	16255	117.976	4,59	241.025	9,66	52,48
	35	17564	557.678	28,75	1.752.401	91,24	68,49
	39	18862	9.266.715	563,59	54.715.192	3527,35	84,02
	43	20173	29.966.621	2435,54	n.d.	n.d.	
ts225	19	21000	3.698	0,03	21.743	0,19	84,21
	23	25000	11.146	0,16	103.669	1,31	87,79
	27	32395	345.338	5,32	4.414.244	69,39	92,33
	31	33395	686.365	20,74	10.573.610	304,26	93,18
	35	36784	3.366.853	149,39	56.651.571	2429,08	93,85
	39	39395	12.705.633	752,21	n.d.	n.d.	
	43	43082	85.718.933	6120,99	n.d.	n.d.	

Table 3: Performance comparison between the *BBL* algorithm and the version without filters.

hour of computing time. In addition, some instances with 51 vertices could be solved within that time limit. This comparison shows that the branch-and-bound algorithm introduced here is outperformed by the branch-and-cut algorithm. However, the implementation of the latter algorithm appears to be much more involved. It also relies on the availability of a powerful solver such as CPLEX. In addition to being easier to implement, the branch-and-bound algorithm introduced here can be easily adapted to handle further restrictions on the construction of tours.

6. Conclusion

This paper has introduced a new branch-and-bound algorithm for the TSPPDL. Following the additive lower bounding paradigm, this algorithm computes lower bounds at each node of the search tree by solving two relaxations: the assignment problem and the shortest spanning r-arborescence problem. Combined with the use of filters to reduce the size of the residual graph, this approach yields an effective algorithm capable of consistently solving instances with up to 35 vertices. In addition, some problems with 39 and 43 vertices have also been solved to optimality.

Acknowledgements

This work was partly supported by the Canadian Natural Science and Engineering Research Council under grant 227837-04. This support is gratefully acknowledged. We are also thankful to three anonymous referees for their valuable comments.

References

R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows*. Prentice Hall, 1993.

P.M. Camerini, L. Fratta, and F. Maffioli. A note on finding optimum branchings. *Networks*, 9:309–312, 1979.

G. Carpaneto, M. Fischetti, and P. Toth. New lower bounds for the symmetric travelling salesman problem. *Mathematical Programming (Series B)*, 45:233–254, 1989.

F. Carrabs, J.-F. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. To appear in *INFORMS Journal of Computing*, 2007.

L. Cassani. Algoritmi euristici per il TSP with rear-loading. Degree Thesis, Università di Milano, Italy. http://www.crema.unimi.it/˜righini/Papers/Cassani.pdf", 2004.

Y.J. Chu and T.H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:13961400, 1965.

J.-F. Cordeau, M. Iori, G. Laporte, and J.-J. Salazar-González. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Technical Report CIRRELT-2007-13, Centre for Research on Transportation, HEC Montréal, 2007a.

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, Volume 14, pages 429–466. Elsevier, Amsterdam, 2007b.

J. Edmonds. Optimum branching. *Journal Research of the National Bureau of Standards*, 71B:233–240, 1967.

F. Ficarelli. Mathematical programming algorithms for the TSP with rear-loading. Degree Thesis, Università di Milano, Italy. http://optlab.dti.unimi.it/Papers/Ficarelli.pdf, 2005.

M. Fischetti and P. Toth. An additive bounding procedure for combinatorial optimization problems. *Operations Research*, 37:319–328, 1989.

M. Fischetti and P. Toth. An additive bounding procedure for the asymmetric travelling salesman problem. *Mathematical Programming*, 53:173–197, 1992.

M. Fischetti and P. Toth. An efficient algorithms for the min-sum arborescence problem on complete digraph. *ORSA Journal on Computing*, 5:4:426–434, 1993.

H. Gabow, Z. Galil, T. Spencer, and R.E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:2:109–122, 1986.

H.N. Gabow, Z. Galil, and T.H. Spencer. Efficient implementation of graph algorithms using contraction. *Journal ACM*, 36:3:540–572, 1989.

G. Gutin and A.P. Punnen. *The Traveling Salesman Problem and Its Variations*. Kluwer, Boston, 2002.

P. Healy and R. Moll. A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research*, 83:83–104, 1995.

B. Kalantari, A.V. Hill, and S.R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22: 377–386, 1985.

S.P. Ladany and A. Mehrez. Optimal routing of a single vehicle with loading and unloading constraints. *Trasportation Planning and Technology*, 8:301–306, 1984.

G. Levitin. Organization of computations that enable one to use stack memory optimally. *Soviet Journal of Computer & System Science*, 24:151–159, 1986.

G. Levitin and R. Abezgaouz. Optimal routing of multiple-load AGV subject to LIFO loading constraints. *Computers & Operations Research*, 30:397–410, 2003.

J.D.C. Little, K.G. Murty, D.W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11:972–989, 1963.

I. Or. *Traveling salesman type combinatorial problems and their relations to the logistics of blood banking*. PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, 1976.

J.A. Pacheco. *Problemas de rutas con ventanas de tiempo*. PhD thesis, Department of Estadistica and Investigation Operativa, Universitad complutense de Madrid, 1994.

J.A. Pacheco. Problemas de rutas con carga y descarga en sistemas LIFO: solutiones exactas. *Estudios de Econimia Aplicada*, 3:69–86, 1995.

J.A. Pacheco. Heuristico para los problemas de ruta con carga y descarga en sistemas LIFO. *SORT, Statistics and Operations Research Transactions*, 21:69–86, 1997a.

J.A. Pacheco. Metaheuristic based on a simulated annealing process for one vehicle pick-up and delivery problem in LIFO unloading systems. In *Proceedings of the Tenth Meeting of the Europen Chapter of Combinatorial Optimization (ECCO X)*. Tenerife, Spain, 1997b.

J. Renaud, F.F. Boctor, and G. Laporte. Perturbation heuristics for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 29:1129–1141, 2002.

J. Renaud, F.F. Boctor, and J. Ouenniche. A heuristic for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 27:905–916, 2000.

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

M.W.P. Savelsbergh. An efficient implementation of local search algorithms for contrained routing problems. *European Journal of Operational Research*, 47:75–85, 1990.

R.E. Tarjan. Finding optimum branching. *Networks*, 7:25–35, 1977.

S.G. Volchenkov. Organization of computations utilizing stack storage. *Engineering Cybernetics, Soviet Journal of Computer & System Science*, 20:109–115, 1982.