New Refinements for the Solution of Vehicle Routing Problems with Branch and Price

DOMINIQUE FEILLET

Laboratoire d'Informatique d'Avignon, 339 Chemin des Meinajariés, BP 1228, 84911 Avignon, France
dominique.feillet@univ-avignon.fr

MICHEL GENDREAU, LOUIS-MARTIN ROUSSEAU

CIRRELT, Université de Montréal, C.P. 6128, succursale centre-ville, Montréal, Canada H3C 3J7
{michel.gendreau,louis-martin.rousseau}@cirrelt.ca

Abstract

Column generation is a well-known mathematical programming technique based on two components: a master problem, which selects optimal columns (variables) in a restricted pool of columns, and a subproblem that feeds this pool with potentially good columns until an optimality criterion is met. Embedded in Branch and Price algorithms, this solution approach proved to be very efficient in the context of numerous vehicle routing problems, where columns represent feasible vehicle routes. The subproblem is then usually expressed as a shortest path problem with resource constraints, which can be solved using dynamic programming methods that are generally very effective in practice. In this paper, we propose some new refinements to improve the capabilities of column generation approaches in this context, with a focus on the subproblem phase. For the sake of simplicity, we restrict our study to the case of the Vehicle Routing Problem with Time Windows. We first introduce the notion of Limited Discrepancy Search, which is well known in the field of Constraint Programming, and we show how LDS can be applied to dynamic programming. We also discuss how the state graph of dynamic programming can be manipulated in order to simulate local search during label extension. Finally, we present some lower bounds that allow removing a substantial number of labels during the search. Computational results demonstrate the considerable impact of these refinements in terms of computing time.

Keywords: column generation; branch and price; vehicle routing.

Introduction

Vehicle routing problems are widely present in today's industries, ranging from distribution problems to fleet management. They account for a significant portion of the operational costs of many companies. Operations research techniques have been used with success in many situations for reducing such costs. Even if most real instances of vehicle routing problems are solved with heuristic methods, the desire to produce optimal solutions has given rise to a prolific research area. Through the years, the Branch and Price methodology proved to be a cornerstone for the exact solution of many vehicle routing problems. This includes routing problems with time windows (Desrochers *et al.* 1992), backhauls (Gélinas *et al.* 1995) or pick-up and delivery (Sol 1994) to mention only a few.

Among these problems, the Vehicle Routing Problem with Time Windows (VRPTW) can be described as follows. Given a set of customers, a set of vehicles, and a depot, the VRPTW is to find a set of routes of minimal total length, starting and ending at the depot, such that each customer is visited by exactly one vehicle to satisfy a specific demand. The time at which a vehicle visits a customer must respect that customer's requested time window. A vehicle can wait in case of early arrival, but late arrival is not allowed. In connection with customer demands, a capacity constraint restricts the load that can be carried by a vehicle.

In 1992, Desrochers *et al.* (1992) published a seminal paper for the solution of the VRPTW with a Branch and Price procedure. The procedure was evaluated on a large set of instances, involving up to 100 customers, but only managed to solve a limited subset of these instances. Since then, several authors have adapted this approach, introducing some more advanced concepts and solving more and more instances. Even so, some of these instances still remain open today while many can only be solved with unduly long computing times.

The purpose of this paper is to propose some new refinements to the Branch and Price methodology, helping in the solution of difficult instances with rather simple techniques. For the sake of simplicity, we restrict our study to the case of the VRPTW. Even so, these refinements could easily be transposed to solve several other vehicle routing problems.

This paper is organized as follows. Section 1 introduces more precisely the VRPTW and reviews Branch and Price approaches for its solution. Section 2 describes the refinements and the algorithm that we propose. Thorough computational experiments first evaluating the algorithm and then the different refinements separately are provided in Section 3. The conclusion follows.

1 Branch and Price Methodology for the VRPTW

In this section we first describe the VRPTW and recall a standard formulation. We then explain the basics of Branch and Price applied to this problem and how this general scheme can efficiently be improved.

1.1 Formal Description of the VRPTW

The VRPTW is defined on a network $G = (V, A)$, where $V = \{v_0, \ldots, v_n\}$ is the set of nodes and A is the set of arcs. Vertex v_0 is a special node called the depot, vertices v_1 to v_n represent customers. A cost c_{ij} and a travel time t_{ij} are defined for every arc $(v_i, v_j) \in A$. Every customer $v_i \in V \setminus \{v_0\}$ has a positive demand d_i, a time window $[a_i, b_i]$ and a positive service time s_i. A fleet of K vehicles of capacity Q is available for servicing the customers. Vehicles must begin and end their routes at the depot within a time horizon $[a_0, b_0]$. The total demand of customers visited by a route is limited by Q. The service of a customer has to start within its time window, but a vehicle is allowed to arrive earlier and to wait. The VRPTW consists in finding a minimum cost set of routes visiting exactly once each customer, while respecting the capacity, time window and fleet size constraints discussed above.

In the following, we make these additional common assumptions: the cost and travel time matrices are supposed to be identical to each other, to be nonnegative, and to satisfy the triangle inequality. For the sake of simplicity, we also define $d_0 = 0$ and $s_0 = 0$.

The VRPTW can then be described with the following model:

$$\text{minimize} \sum_{1 \leq k \leq K} \sum_{(v_i, v_j) \in A} c_{ij} f_{ij}^k \tag{1}$$

subject to

$$\sum_{\{v_j \in V | (v_i, v_j) \in A\}} f_{ij}^k - \sum_{\{v_j \in V | (v_j, v_i) \in A\}} f_{ji}^k = 0 \qquad (v_i \in V, 1 \leq k \leq K), \tag{2}$$

$$\sum_{\{v_i \in V | (v_0, v_i) \in A\}} f_{0i}^k \leq 1 \qquad (1 \leq k \leq K), \tag{3}$$

$$\sum_{1 \leq k \leq K} \sum_{\{v_j \in V | (v_i, v_j) \in A\}} f_{ij}^k = 1 \qquad (v_i \in V \setminus \{v_0\}), \tag{4}$$

$$\sum_{(v_i, v_j) \in A} d_i f_{ij}^k \leq Q \qquad (1 \leq k \leq K), \tag{5}$$

$$s_i^k + s_i + c_{ij} - s_j^k + M f_{ij}^k \leq M \qquad ((v_i, v_j) \in A, v_j \neq v_0, 1 \leq k \leq K), \tag{6}$$

$$s_i^k + s_i + c_{i0} - b_0 + M f_{i0}^k \leq M \qquad ((v_i, v_0) \in A, 1 \leq k \leq K), \tag{7}$$

$$a_i \leq s_i^k \leq b_i \qquad (v_i \in V, 1 \leq k \leq K), \tag{8}$$

$$f_{ij}^k \in \{0, 1\} \qquad ((v_i, v_j) \in A, 1 \leq k \leq K), \tag{9}$$

where f_{ij}^k and s_i^k are decision variables and M is a large number. Variable f_{ij}^k indicates whether arc (v_i, v_j) is used by vehicle k or not. For a customer v_i visited by a vehicle k^*, $s_i^{k^*}$ is the time at which service starts for v_i, while s_i^k is meaningless for $k \neq k^*$. For the depot, s_0^k is the departure time of vehicle k.

Constraints (2)-(3) define the route structure for the vehicles. Constraints (4) enforce the visit of every customer. Constraints (5) and constraints (6)-(8) respectively concern vehicle capacity and time windows.

1.2 Branch and Price Methodology for the VRPTW

In this subsection, we describe the principles of Branch and Price algorithms for the VRPTW and we review the most important work on the subject.

The motivation for using a Branch and Price technique here is that the linear relaxation of model (1)-(9) is very weak. Hence, this model cannot be used directly with a Branch and Bound approach, except when fairly small instances are considered. To circumvent this difficulty, Branch and Price methods rely on a different model having a better linear relaxation. This new model can be obtained through a Dantzig-Wolfe decomposition from (1)-(9) as detailed in Desrochers *et al.* (1992). Before presenting this model, we have to introduce some new notation. Let $\Omega = \{r_1, \ldots, r_{|\Omega|}\}$ be the set of feasible vehicle routes, i.e., the set of paths in G issued from the depot, going to the depot, satisfying capacity and time window constraints and visiting at most once each customer. Let c_k be the cost of route $r_k \in \Omega$. Let $a_{ik} = 1$ if route $r_k \in \Omega$ visits customer v_i and 0 otherwise. The new model for the VRPTW is then:

$$\text{minimize} \sum_{r_k \in \Omega} c_k x_k \tag{10}$$

subject to

$$\sum_{r_k \in \Omega} a_{ik} x_k \geq 1 \qquad (v_i \in V \setminus \{v_0\}), \tag{11}$$

$$\sum_{r_k \in \Omega} x_k \leq K, \tag{12}$$

$$x_k \in \{0, 1\} \qquad (r_k \in \Omega). \tag{13}$$

In this model, decision variable x_k indicates whether a route r_k is used in the solution or not. Constraint (12) limits the number of vehicles used. Constraints (11) enforce that each customer is visited at least once. Note that these constraints are not formulated as $\sum_{r_k \in \Omega} a_{ik} x_k = 1$ for technical reasons to be explained later and because the triangle inequality ensures that there exists an optimal solution of (10)-(13) visiting each customer exactly once.

Although having a better linear relaxation value than (1)-(9), this model is not tractable with a standard Branch and Bound approach. This stems from the fact that the size of the set Ω grows exponentially with instance size. Indeed, the linear programs that would be used to evaluate search tree nodes would contain too many variables to be solved in a classical manner. This evaluation can, however, be tackled with a column generation technique, which, when repeated at every node of the Branch and Bound tree, yields the so-called Branch and Price algorithm.

Column generation can be described as follows. We call the Master Problem (MP) the linear relaxation of (10)-(13). We introduce $MP(\Omega_1)$, the restriction of the Master Problem MP to a subset of variables

$\Omega_1 \subset \Omega$. $MP(\Omega_1)$ is called the Restricted Master Problem. Let also $D(\Omega_1)$ be the dual program of $MP(\Omega_1)$. Note that MP is then identically $MP(\Omega)$ and that $D(\Omega)$ is the dual program of MP. The optimal solution of $MP(\Omega_1)$, with the simplex algorithm for instance, provides an optimal solution λ^* for $D(\Omega_1)$. This solution is also a solution of $D(\Omega)$, but it is not necessarily feasible. When every dual constraint deriving from the routes in $\Omega \setminus \Omega_1$ is satisfied, the solution λ^* is feasible for $D(\Omega)$, and therefore optimal, since $D(\Omega)$ is more constrained than $D(\Omega_1)$. When one or several constraints deriving from the routes in $\Omega \setminus \Omega_1$ are violated, the principle of the column generation method is to identify one or several of these constraints, with the help of a subproblem, in order to integrate the corresponding variables in the set Ω_1. Thus, solving alternately $MP(\Omega_1)$ and the subproblem allows to converge toward dual feasibility. The algorithm terminates when the subproblem solution attests that there are no more violated constraints and that therefore the current dual solution is feasible for $D(\Omega)$.

In our situation, dual constraints are of the form $\sum_{v_i \in V \setminus \{v_0\}} a_i^k \lambda_i + \lambda_0 \leq c_k$, where λ_i is the nonnegative dual variable associated with the visit of customer v_i (constraints (11)) and λ_0 is the nonpositive dual variable associated with the fleet size constraint (12). Note that formulating constraints (11) $\sum_{r_k \in \Omega} a_{ik} x_k = 1$ would have led to free variables λ_i, which would have complicated the convergence of the algorithm.

The purpose of the subproblem is finally to find routes $r_k \in \Omega$ such that

$$c_k - \sum_{v_i \in V \setminus \{v_0\}} a_i^k \lambda_i - \lambda_0 < 0.$$

It consists equivalently of columns with a negative reduced cost in $MP(\Omega)$, when the basic solution is the optimal solution of $MP(\Omega_1)$. In the following, we call these columns routes with a negative reduced cost. Using the notation $\delta_{ij}^k = 1$ when (v_i, v_j) is included in r_k and $\delta_{ij}^k = 0$ otherwise, this condition can be expressed as:

$$\sum_{(v_i, v_j) \in A} \delta_{ij}^k (c_{ij} - \lambda_i) < 0.$$

From the above expression and from the definition of a vehicle route, we see that the subproblem reduces to an elementary shortest path problem with resource constraints (ESPPRC) from the depot to the depot, satisfying capacity and time constraints, where the cost of each arc (v_i, v_j) is $(c_{ij} - \lambda_i)$. A solution procedure based on dynamic programming for this problem is proposed in Feillet *et al.* (2004). Dynamic programming is particularly well adapted to this context because it computes a set of Pareto optimal paths and might provide MP with several columns at a time. Here, we give a brief description of the algorithm used to solve the subproblem.

The algorithm is an extension of the classical Bellman's algorithm. The principle is to construct partial paths, that are successively extended in every direction checking resource constraints. The mechanism is initiated with a void path corresponding to the starting depot and is stopped when every possible extension has been performed for every partial path appeared during the process. Dominance rules are used to compare partial paths arriving at a same location and to discard some of them. Feasibility and dominance rules lead us to characterize partial paths by labels of the form $L = (L_c, L_t, L_l, L_u, L_e)$, with fields respectively representing the cost, the shortest service starting time at the ending vertex, the load level, the set of unreachable vertices, and the ending vertex of the partial path. Unreachable vertices are vertices that cannot be reached anymore due to resource constraints or because they already have been visited. This information is used to maintain elementary paths and is coded with a set of binary resources, indicating the reachability status for every vertex. Unlike Bellman's algorithm when no resources are considered, each vertex of the graph can maintain a large number of labels since the comparison of two labels takes into account their consumption level for each resource. A common practice is to avoid computing the complete Pareto optimal path set by stopping the subproblem solution algorithm prematurely when a sufficient number of good columns has been found.

Another important point of the Branch and Price method concerns the branching scheme. Instead of branching on MP variables, it is generally preferred to branch on variables f_{ij}^k from the original formulation (1)-(9). It is fairly easy to see that when MP has a fractional optimal solution, there exists at least one arc (v_i, v_j) that is traversed a fractional number of times f_{ij}, where $f_{ij} = \sum_{r_k \in \Omega_1} \delta_{ij}^k x_k$ and with $0 < f_{ij} < 1$. It is then possible to derive two branches: one branch where v_j cannot follow v_i, the other where a vehicle visiting v_i necessarily goes immediately to v_j. These two rules can very easily be transposed to MP: in both cases, inadequate columns are just set to 0. For the subproblem, it is not much more complicated. In the first case, arc (v_i, v_j) is simply removed. In the second case, any arc (v_i, v_l) with $v_l \neq v_j$ and any arc (v_l, v_j) with $v_l \neq v_i$ must be removed.

1.3 Advanced Branch and Price Implementations

In this subsection we review several techniques that have been introduced within this general scheme and that have significantly increased its efficiency.

In the first implementation of Branch and Price for the VRPTW, Desrochers *et al.* (1992) transformed (10)-(13) to obtain a more tractable subproblem. This transformation takes advantage of the fact that the Elementary Shortest Path Problem with Resource Constraints, though NP-hard in the strong sense, admits a pseudo-polynomial algorithm when the elementary path condition is removed. Thereby, Desrochers *et al.* (1992) propose the two following modifications:

- Ω is enlarged to include non-elementary paths,
- a_i^k becomes the number of times vertex v_i is visited in route r_k.

With these two simple changes, the model remains valid and the subproblem is changed into a (non-elementary) shortest path problem with resource constraints (SPPRC). It can be solved with a dynamic programming procedure similar to the one mentioned above. Actually, Feillet *et al.*'s algorithm (2004) for the ESPPRC derives from this one. Even if the size of the subproblem state space increases with this modification, dominance rules are far more efficient and the global efficiency is improved. However, with this new formulation, the linear relaxation provides a weaker lower bound, which complicates the pruning of nodes during the tree search.

Actually, Desrochers *et al.* (1992) were more clever and forbade paths with 2-cycles, i.e., paths with cycles composed of two arcs. This condition can easily be added to the Master Problem and can be handled by the SPPRC dynamic programming solution algorithm with a low computing cost. Following this idea, Irnich (2001) and Irnich and Villeneuve (2003) study the removing of k-cycles. They evaluate it for $k = 3$ and $k = 4$ and conclude that the quality of the lower bound can come significantly closer to the ESPPRC-based bound, with a reasonable computing time. Note that in parallel with the work of of Feillet *et al.* (2004), Chabrier (2006) adapted the SPPRC dynamic programming solution algorithm to effectively address the ESPPRC situation. This collection of ideas has recently converged toward an efficient compromise solution simultaneously developed by Boland *et al.* (2006) and Righini and Salani (2005). These authors propose to solve dynamically the ESPPRC by progressively adding the elementary path constraint. The SPPRC is first solved. If the set of routes of negative cost found is not empty and only contains non-elementary routes, single-visit constraints are added for some vertices and the problem is solved again. This process stops when an elementary route of negative cost is found or when no route (elementary or not) with negative reduced cost exists.

Another approach proposed for increasing the quality of the lower bound is the addition of cuts. Kohl *et al.* (1999) introduced the so-called k-path cuts:

$$\sum_{v_i \in V \setminus S} \sum_{v_j \in S} f_{ij} \geq k, \tag{14}$$

where f_{ij} is the flow on arc (v_i, v_j) as defined above and S is a vertex set whose demand cannot be satisfied with $k - 1$ vehicles. These cuts can easily be integrated into the Master Problem and handled

by the subproblem by only taking into account new dual variables on appropriate arcs. However, they are quite complicated to separate. Kohl *et al.* (1999) limit their study to k-path cuts with $k \leq 2$. The case $k = 1$ is very simple. When $k = 2$, a heuristic algorithm is proposed to find maximal sets S such that $\sum_{v_i \in V \setminus S} \sum_{v_j \in S} f_{ij} < 2$ and a Traveling Salesman Problem with Time Windows (TSPTW) solution algorithm is used to determine whether these sets can be visited with a single vehicle. The case $k = 3$ was addressed later by Cook and Rich (1999). These cuts were recently generalized in Desaulniers *et al.* (2006). Other types of cuts, called subset row inequalities and based on the Chvatal-Gomory cutting scheme, were also recently proposed in Jespen *et al.* (2006).

Finally, Kallehauge *et al.* (2001) proposed to hybridize the Branch and Price scheme with Lagrangean relaxation. Column generation can then be viewed as a decomposition scheme where the Master Problem and the subproblem communicate with the help of values λ_i, when these values are set to be the dual variables of the Master Problem. Lagrangean relaxation involves an identical Master Problem and subproblem, communicating with values λ_i, but these values are guided by a different strategy. Kallehauge *et al.* (2001) propose to use Lagrangean relaxation at the root node to improve the convergence of the method.

Beside these strategies, many computational tricks have been proposed to accelerate the solution process. Most of these tricks are summarized in Desaulniers *et al.* (2001). These tricks have enabled a large set of instances to be solved for the first time. The recent working paper by Desaulniers *et al.* (2006) integrates most of these ideas. The resulting algorithm is rather complex but provides the most efficient approach up to now.

2 New Acceleration Techniques

The contribution of this paper lies in the following improvements that are incorporated in the subproblem solution algorithm. Note that contrary to usual implementations of column generation for routing problems, we only consider elementary routes in Ω, i.e., routes where customers are never visited more than once. While these techniques are implemented in the context of elementary shortest paths, they can easily be transposed to a non-elementary context, unless the opposite is explicitly mentioned.

2.1 Limited Discrepancy Search

Limited Discrepancy Search (LDS) is a well-known tree search method, introduced by Harvey and Ginsberg (1995) in the context of Constraint Programming (CP). A heuristic criterion is used, indicating which descendant nodes of a given node are the most promising. A branching decision that does not lead the search towards these nodes is called a discrepancy. An upper bound on the number of discrepancies limits the search: a condition to explore a node is that the number of discrepancies accumulated along the path connecting the root node to this node does not exceed this upper bound. The search is thus limited to the most promising part of the arborescence, according to the heuristic criterion. However, compared to a heuristic tree search that would only explore good branches, LDS has the advantage of allowing some rare bad decisions during the search, which might reflect the structure of optimal solutions. While a satisfactory solution is not found (generally a feasible solution in the context of CP), the search is repeated with an increasing discrepancy limit, possibly until the search is complete.

We embed the concept of LDS in the Dynamic Programming (DP) solution scheme described in Section 1. Our objective is to efficiently drive the search towards the most promising paths, i.e., quickly generate paths of negative value. For each vertex of the original VRPTW, we need to partition the set of neighbor nodes into two sets: a set of good neighbors and a set of bad ones. These sets are constructed according to reduced cost values, as detailed below. LDS is then implemented by adding an additional field L_d to the labels. This new field indicates for each label the number of discrepancies performed to generate the label: $L_d = 0$ for the label initiating the DP algorithm at the depot node; L_d is incremented when L is extended

in direction of one of its bad neighbors. If the extension of a label would cause the discrepancy level of the new label to exceed the current discrepancy limit, this extension is discarded.

Figure 1 provides an illustration of this concept. In this figure, bad neighbors are represented with dashed arcs. With a discrepancy limit 0, the only labels accepted are the ones propagating along the path $v_0 \rightarrow v_1 \rightarrow v_4 \rightarrow v_5$. When the discrepancy limit is 1, all labels with ending vertex v_1, v_3 or v_4 are accepted; labels with ending vertex v_2 or v_5 are accepted except $v_0 \rightarrow v_3 \rightarrow v_2$ and $v_0 \rightarrow v_3 \rightarrow v_2 \rightarrow v_5$; vertex v_6 can only be reached with label $v_0 \rightarrow v_6$. With a discrepancy limit 2, every label is accepted except $v_0 \rightarrow v_3 \rightarrow v_2 \rightarrow v_6$. The search is complete when the discrepancy limit is 3.

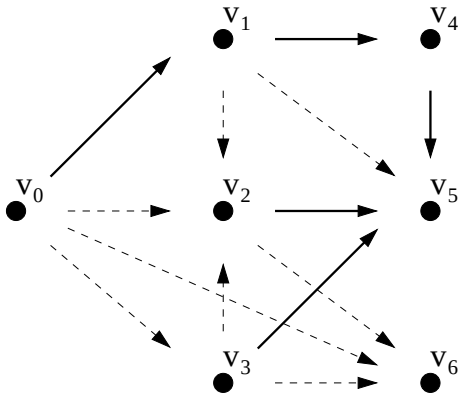

Figure 1: LDS in Dynamic Programming.

Although simple, this technique can be implemented in many ways. In our implementation, we have proceeded as follows. Two parameters are defined for the size of the good neighbor set and for the number of discrepancies allowed (called DISC). The size of the good neighbor set is fixed throughout the solution process to two. DISC is increased when the subproblem fails to find new routes, which we refer to as a *fail*. The ESPPRC is then solved again until routes are found or the value of DISC ensures that the search is complete, which closes the subproblem solution phase.

Good neighbors are the two reachable neighbors with the minimum reduced cost value $c_{ij} - \lambda_i$, plus the depot. For the sake of efficiency, lists of successors are sorted according to reduced cost at each new iteration of the column generation scheme (i.e., when dual values have changed). This allows us to stop trying to extend a label towards its neighbors once an extension is discarded due to discrepancy.

The DISC parameter is regulated as follows. DISC is initialized at 0, which implies that the subproblem is first solved using only good neighbors. After a fail, the DISC parameter is generally increased by 1. When the new value exceeds 25% of the maximal number of customers that a route can contain (computed as explained below), DISC is fixed to this latter value, so that the search is complete. If DISC is maximal and the subproblem still fails to find new routes, column generation is stopped. When, following a fail, the ESPPRC is solved again with a higher value for DISC, former labels are preserved. DP then performs normally with a set of labels that can potentially be extended.

This method quickly finds routes that mainly connect customers to their nearest neighbours, but use a short number of long arcs, where "nearest" and "long" are measured with respect to reduced cost value. By contrast, a classic heuristic tree search that progressively enlarges the set of neighbors, as described in Desaulniers *et al.* (2001), will have difficulties in finding such routes.

The upper bound on the maximal number of vertices in routes is precomputed with the solution of two knapsack problems where items are vertices and weights are respectively the demand and the minimum time consumed for visiting a vertex (service time plus cost of the cheapest outgoing arc). In both cases,

the item corresponding to the depot is enforced in the solution. These two knapsack problems are:

$$\text{maximize } z^1_{KP} = 1 + \sum_{v_i \in V \setminus \{v_0\}} y_i$$

subject to

$$\begin{aligned} \sum_{v_i \in V \setminus \{v_0\}} d_i y_i &\leq Q, \\ y_i \in \{0,1\} \qquad & (v_i \in V \setminus \{v_0\}). \end{aligned}$$

and

$$\text{maximize } z^2_{KP} = 1 + \sum_{v_i \in V \setminus \{v_0\}} y_i$$

subject to

$$\begin{aligned} \min_{v_j \in V \setminus \{v_0\}} c_{0j} + \sum_{v_i \in V \setminus \{v_0\}} (s_i + \min_{v_j \in V \setminus \{v_i\}} c_{ij}) y_i &\leq b_0 - a_0, \\ y_i \in \{0,1\} \qquad & (v_i \in V \setminus \{v_0\}). \end{aligned}$$

The upper bound is then given by $\min\{z^1_{KP}, z^2_{KP}\}$. As all items have unit costs, the two knapsack problems can be solved quickly with the smallest-weight-first rule.

Note that more precise estimations of the maximal number of vertices in routes could be performed, but would not be very useful here, since this estimation is only used for triggering LDS off.

2.2 Label Loading and Meta Extensions

The motivation behind these techniques is to use, whenever possible, the information about the "good" paths that have been previously identified. Indeed, once MP is solved, we already have in hand a number of routes whose reduced cost value is zero (namely, MP basic columns).

Label Loading (LL) consists of adding a set of labels to the graph before the DP search process is undertaken. This is very simple and has presumably been implemented in other DP algorithms addressing similar problems. Our implementation consists in selecting in Ω_1 all routes r for which $x_r > 0$. We then traverse each of these routes while generating the label associated with the visit to each vertex. All the labels thus generated are added to the DP graph before we start the solution process. Label Loading is illustrated in Figure 2. In this figure, $L_{i_1-\ldots-i_k}$ represents the label associated with partial path $\{v_{i_1}, \ldots, v_{i_k}\}$

The Meta Extension (ME) operator is used to obtain the complementary effect of Label Loading. While traversing each of the routes previously selected, we also add new *metavertices* to the original graph. These metavertices correspond to the remaining path from their associated original vertex to the destination depot and can be viewed as *metadepots*. Every metavertex is defined as a successor of its associated original vertex. For example, when traversing route $\{v_0, v_1, v_3, v_4, v_0\}$, the metavertex we associate with v_3 is equivalent to the remaining subpath $\{v_3, v_4, v_0\}$. This metavertex is added in the graph as a successor of v_3, as illustrated in Figure 3.

When a metavertex is added to the graph, we compute upper bounds on cost, time and load that would allow any path to be extended along the remaining partial path represented by the metavertex, to obtain a time- and capacity-feasible negative reduced cost route. This is easily done with a backward strategy from the final depot. To be extended to a metavertex, a label must respect these bounds. When this happens, we can obtain a complete route by simply checking that the elementary path condition holds (i.e., that no vertex is visited more than once). The resulting label is then added to the set of depot labels as any other label extended to the depot.

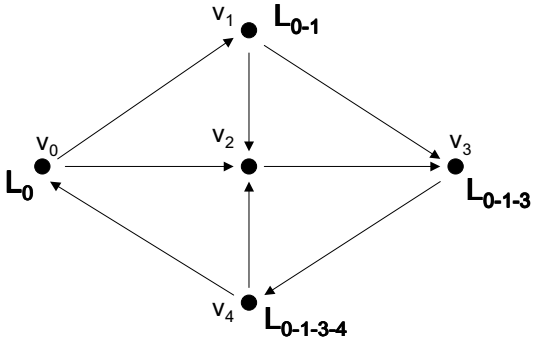

Figure 2: Label Loading: initial DP graph when $x_r > 0$ for route $r = \{v_0, v_1, v_3, v_4, v_0\}$.

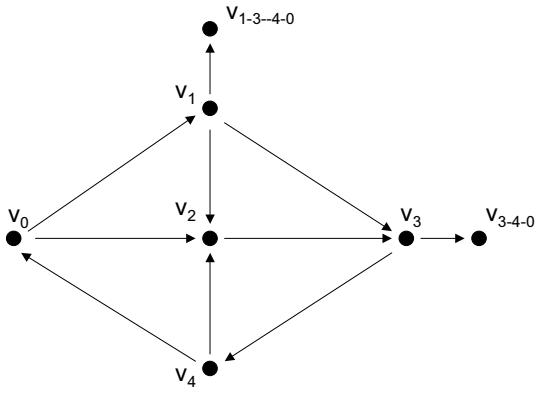

Figure 3: Meta Extension: metavertices when $x_r > 0$ for route $r = \{v_0, v_1, v_3, v_4, v_0\}$.

Note that, with regard to the LDS policy, metavertices are considered as good neighbors, but are not counted in the good neighbor set.

The two techniques LL and ME, when used together, have the ability to rapidly identify small variants of the current MP solution routes. Operators like node insertion, node deletion, and path crossing (connecting the end of one route to the beginning of another) can be obtained with only a few label extensions. This enables rapid exploration of the neighborhood of the current MP solution. These techniques are very helpful to guide the search with the help of the global information provided by the current MP solution. They are very complementary with the LDS approach, which drives the search using local information (arc reduced costs). Actually, one can see the combination of these two techniques as a method progressively switching from local search to dynamic programming. It is also interesting to note that ME can be seen as a way of capturing the benefits of bi-directional DP for the ESPPRC, which was shown to be quite effective by Righini and Salani (2006).

2.3 Label Elimination

When the resource limits are not very constraining, a very large number of labels can be generated during the search for negative reduced costs paths. Many of these labels correspond to ineffective combinations

of arcs and none of their extensions is going to provide a valuable column. The motivation here is to determine (and remove) such labels, that can never be extended to the final depot with a negative cost.

What we propose is to compute two very simple lower bounds on the costs of all paths that can be generated from a given label $L = (L_c, L_t, L_l, U, v_k)$. The principle is to determine whether extensions of L may yield a negative cost path or not. The bounding schemes have to be chosen to balance the quality of the information they provide with the time taken for computation. The two bounding schemes that we propose are very similar to the ones presented in Subsection 2.1 and rely on simple knapsack problems defined as follows:

$$\text{minimize } L_c + (\min_{v_j \in V \setminus \{v_0\}} c_{j0} - \lambda_0) + \sum_{v_i \in V \setminus (U \cup \{v_0\})} (\min_{v_j \in V \setminus \{v_i\}} c_{ji} - \lambda_i) y_i$$

subject to

$$\sum_{v_i \in V \setminus (U \cup \{v_0\})} d_i y_i \leq Q - L_l,$$
$$y_i \in \{0,1\} \qquad (v_i \in V \setminus (U \cup \{v_0\})).$$

and

$$\text{minimize } L_c + (\min_{v_j \in V \setminus \{v_0\}} c_{j0} - \lambda_0) + \sum_{v_i \in V \setminus (U \cup \{v_0\})} (\min_{v_j \in V \setminus \{v_i\}} c_{ji} - \lambda_i) y_i$$

subject to

$$\min_{v_j \in V \setminus \{v_0\}} c_{j0} + \sum_{v_i \in V \setminus (U \cup \{v_0\})} (s_i + \min_{v_j \in V \setminus \{v_i\}} c_{ji}) y_i \leq b_0 - L_t - s_k,$$
$$y_i \in \{0,1\} \qquad (v_i \in V \setminus (U \cup \{v_0\})).$$

In both cases, the set of items to be selected is a subset of customers. This subset is the complete set of customers minus the set U of unreachable customers. Vertex v_0 is automatically selected, since returning to the depot is compulsory. The cost of an item (customer) is the cost of the cheapest ingoing arc minus the dual price of the customer. Note that λ_0 is the dual variable associated with constraint (12) and is counted when the label returns to the depot in our implementation.

The first bounding scheme relies on a load-based constraint. The consumption level of a customer for this constraint is the demand of the customer. The higher limit is the remaining load allowable in the vehicle. The second bounding scheme relies on a time-based constraint. The consumption level of a customer is its service time plus the cost of the cheapest ingoing arc. The higher limit is the remaining time allowable to return to the depot.

For the sake of efficiency, we only compute the linear relaxation of these bounds. This can be done in linear time when items are sorted, which is done once, before dynamic programming is started. The computing time needed is acceptable in the context of an Elementary Shortest Path algorithm where the elementary path condition already necessitates linear computations for constructing a label. If the cost of one of these bounds is nonnegative the label is removed. Note that items with a nonnegative value can directly be discarded. Also, the solution of the knapsack LP relaxations can be stopped as soon as a negative cost is reached. As this technique is expected to be useful essentially when the number of labels is large, we only trigger Label Elimination (LE) when the DISC parameter is at least 2.

Note that our lower bounds would not be valid when the elementary path condition is relaxed: in this case, new lower bounding schemes would be required in order to apply Label Elimination. Furthermore, these schemes should preferably be computed in constant time to avoid slowing down label extension. Hence, contrary to previous techniques (LDS, LL and ME), Label Elimination seems only convenient for

the elementary path version of the column generation scheme. LE can however be integrated in solution schemes where the elementary condition is partially and dynamically introduced (Boland *et al.* 2006, Righini and Salani 2005 – see Section 1); indeed, some labels might be discarded wrongly, but a path of negative cost will always be found if an elementary path of negative cost exists, which ensures the proper functioning of the algorithm.

One should also note that, as we were performing this study, Lübbecke (2005) investigated the use of lower bounds in column generation subproblems when solved by dynamic programming. Our proposal can then be seen as a special case and as an implementation of the general scheme he sketches.

Note finally that when 2-path cuts are used, new dual variables λ_{ij}, issued from constraints (14), should be considered on some arcs. These dual prices have to be subtracted from item costs when computing the bounds defined above, changing $\min_{v_j \in V \setminus \{v_i\}} c_{ji} - \lambda_i$ to $\min_{v_j \in V \setminus \{v_i\}} (c_{ji} - \lambda_{ji}) - \lambda_i$.

2.4 Other Computational Issues

As noted before, 2-path cuts stand out as a standard component of column generation schemes for vehicle routing problems. These cuts are very useful for improving the quality of the bounding scheme and for limiting the number of nodes explored during the tree search. Although less essential when the ESPPRC-based lower bound is used, we have included this component in our implementation. 2-path cuts are generated at the root node of the search tree. They are implemented as described in Kohl *et al.* (1999) except for three points. First, TSPTW feasibility is checked using the ESPPRC solution module. This is simply done by giving high rewards (dual values) to the requested vertices and null rewards to the others. Second, we generate 2-path cuts each time that DISC (the LDS parameter) is greater than two, instead of waiting for the end of the column generation process. By doing this, we avoid having to solve repeatedly the last iterations of the process for which DISC is high and computing times are longer. Third, a limit is set on the total time spent for generating 2-path cuts. This limit is equal to 50% of the current running time. When the total computing time spent for generating 2-path cuts reaches this limit, 2-path cuts are not searched for anymore. This limit avoids wasting a lot of computing time by trying to solve some very difficult TSPTW instances, although some useful cuts may be missed.

It is also well known that column generation methods often show very slow convergence due to heavy degeneracy problems. In order to limit this phenomenon, we use a very simple stabilization method, Interior Point Stabilization, described in Rousseau *et al.* (2007). However, the impact of this method on computing times is rather slight here, since it mainly influences the first iterations of column generation, which are very fast here. Actually, as mentioned above, LDS already acts as a stabilization method that will constraint dual variables through a sometimes large number of quick iterations. As a matter of fact, we stop using Interior Point Stabilization as soon as 2-path cuts are generated or when the tree search begins.

A third noticeable issue concerns the handling of branching. At each node of the search tree, we classically first try to branch on the number of vehicles. Unfortunately, limiting the number of vehicles sometimes leads to infeasibility. We then check whether feasibility can be recovered easily by only generating new columns with the DISC parameter set to 0. We adapt this strategy when the maximal number of vehicles is set to one. Indeed, in this case, the possibility that a solution using a single vehicle exists is very small. We prefer to just check whether the new restricted master problem is feasible or not, without attempting to generate new columns. We branch on the number of vehicles when feasibility is recovered. Otherwise, or when the number of vehicles is not fractional, we select an arc for branching. We then proceed in the following way. The impact of the removal of every arc traversed a fractional number of times is evaluated by solving the updated Restricted Master Problem. The arc whose deletion has the larger impact is selected. This policy enables us to derive two branches for which the new constraint has an effective impact. The time needed for the selection of the arc is needlessly long when subproblems are

very easy to solve, but this time becomes negligible for difficult instances, where limiting the number of nodes in the search tree can be very useful.

Finally, some implementation details need to be mentioned. During the subproblem solution phase, the first 50 columns extended to the destination depot with a negative cost are stored, without referring to dominance rules. Dominance rules are then activated for the remaining columns. This precaution avoids removing good columns when they are rare. The subproblem is stopped as soon as 500 columns of negative cost have been found, even if dominance rules have rejected many of them. We then attempt to complete the set of columns by extending every label of the DP graph towards the depot. At the Master Problem level, integrality of solutions is checked each time the simplex algorithm has been called for. This helps in finding good solutions quickly. The nodes of the search tree are treated in a best-first order. This order is based on the value of the linear relaxation for their parent node. This enables us to maintain an increasing lower bound throughout the algorithm. When a linear program reaches this lower bound, one can then avoid triggering column generation uselessly.

3 Experimental Results

We have evaluated the performance of the proposed techniques on the well-known Solomon instances (1983). These instances are constituted of 3 types of geographical layouts. Customers are randomly located in problem sets r and clustered in problem sets c, while problem sets rc display a mix of random and clustered structures. Each type of instances is divided into two parts, the first part ($r101$-$r112$, $c101$-$c109$, $rc101$-$rc108$) having narrower time windows than the second part ($r201$-$r211$, $c201$-$c208$, $rc201$-$rc208$). Customer coordinates are identical for all instances within one type (i.e., $r1$,...,$rc2$): within one type, instances only differ with respect to the width of the time windows. Each instance contains 100 customers, but smaller instances are created by considering only the first 25 or the first 50 customers. In the data sets, the distance matrix is not explicitly stated, but customer locations are given. Euclidean distances between these customers are calculated with one decimal point and truncation, to allow comparison with other published methods.

The computational study is divided into two parts. First, we evaluate the algorithm and compare its performance with other column generation based solution schemes. Second, we select a representative subset of instances and evaluate more deeply the impact of our refinements.

Computational experiments were carried out on a 1.6 GHz processor with 256 Mb of RAM. The Master Problem was modeled and solved with Cplex 9.0 leaving all parameters to their default values. The maximum allowed time to find a solution was set to 3600 seconds.

3.1 Evaluation of our Algorithm

Tables 1 to 6 show the efficiency of our algorithm. In these tables, *LP bound* and *IP* are respectively the value of the linear relaxation at the root node (rounded to one decimal place) and the value of the optimal solution. Columns *CPU*, *Iter* and *Col* indicate the computing time, the number of subproblem calls and the number of generated columns, respectively for the computations of the linear relaxation and the integer solution (column *CPU* includes the computing time of the linear relaxation for the integer solution). Column *Cuts* gives the number of 2-path cuts generated, while column *Nodes* represents the number of nodes explored in the search tree. When the linear relaxation and/or the optimal integer solution could not be found in the imparted time, an estimation of these values is given in italic. Other columns are left blank to highlight the fact that the instance is not solved. The value provided in the *LP bound* column is the current overestimation of the linear relaxation when the algorithm stops; the value given in the *IP* column is the best solution found so far.

Instance	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r101-025	617.1	0.2	8	73	0	617.1	0.2	8	73	1
r102-025	547.1	0.4	15	170	1	547.1	0.5	15	170	1
r103-025	454.6	0.5	12	194	0	454.6	0.5	12	194	1
r104-025	416.9	0.7	15	226	0	416.9	0.7	15	226	1
r105-025	530.5	0.4	12	148	0	530.5	0.4	12	148	1
r106-025	465.4	1.1	37	232	6	465.4	1.1	37	232	1
r107-025	424.3	0.8	16	203	0	424.3	0.9	16	203	1
r108-025	397.3	2.7	35	302	3	397.3	2.8	35	302	1
r109-025	441.3	1.0	11	115	0	441.3	1.1	11	115	1
r110-025	438.8	1.3	29	263	3	444.1	2.7	64	387	7
r111-025	428.8	0.9	22	182	3	428.8	0.9	22	182	1
r112-025	387.9	2.2	23	418	4	393.0	13.5	108	1147	24
c101-025	191.3	0.3	8	260	0	191.3	0.4	8	260	1
c102-025	190.3	1.2	15	594	0	190.3	1.2	15	594	1
c103-025	190.3	1.4	13	613	0	190.3	1.4	13	613	1
c104-025	186.9	8.5	29	779	0	186.9	8.5	29	779	1
c105-025	191.3	0.6	12	288	0	191.3	0.6	12	288	1
c106-025	191.3	0.4	9	281	0	191.3	0.4	9	281	1
c107-025	191.3	0.4	8	212	0	191.3	0.4	8	212	1
c108-025	191.3	0.4	9	142	0	191.3	0.4	9	142	1
c109-025	191.3	1.2	17	412	0	191.3	1.3	17	412	1
rc101-025	461.1	1.0	33	334	8	461.1	1.0	33	334	1
rc102-025	351.8	0.4	6	178	0	351.8	0.4	6	178	1
rc103-025	332.8	0.8	12	185	0	332.8	0.8	12	185	1
rc104-025	306.6	0.5	8	119	0	306.6	0.5	8	119	1
rc105-025	411.3	0.6	15	210	0	411.3	0.7	15	210	1
rc106-025	345.5	0.7	16	208	0	345.5	0.8	16	208	1
rc107-025	298.3	0.8	14	249	0	298.3	0.8	14	249	1
rc108-025	294.5	1.6	11	240	0	294.5	1.7	11	240	1

Table 1: Optimal solutions Series 1: 25 customers

Instance	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r201-025	460.1	0.6	14	231	0	463.3	1.3	44	502	3
r202-025	410.5	0.6	11	236	0	410.5	0.7	11	236	1
r203-025	391.4	3.0	20	524	0	391.4	3.1	20	524	1
r204-025	350.5	13.6	28	772	0	355.0	37.1	136	1586	11
r205-025	390.6	2.5	26	395	0	393.0	3.8	62	518	3
r206-025	373.6	1.7	17	446	0	374.4	4.6	46	564	5
r207-025	360.1	4.6	17	828	0	361.6	18.0	81	1223	7
r208-025	328.2	18.4	24	1183	0	328.2	18.4	24	1183	1
r209-025	364.1	2.7	26	598	0	370.7	8.6	115	1251	9
r210-025	404.2	2.5	19	412	0	404.6	5.8	77	1130	3
r211-025	341.3	5.0	26	794	0	350.9	98.3	400	3441	42
c201-025	214.7	0.5	12	201	0	214.7	0.5	12	201	1
c202-025	214.7	1.6	15	471	0	214.7	1.7	15	471	1
c203-025	214.7	8.4	21	889	0	214.7	8.4	21	889	1
c204-025	213.1	540.2	48	1013	0	213.1	540.2	48	1013	1
c205-025	214.7	0.7	12	409	0	214.7	0.7	12	409	1
c206-025	214.7	0.8	12	561	0	214.7	0.8	12	561	1
c207-025	214.5	1.4	12	576	0	214.5	1.4	12	576	1
c208-025	214.5	0.8	12	343	0	214.5	0.8	12	343	1
rc201-025	360.2	0.4	10	238	0	360.2	0.4	10	238	1
rc202-025	338.0	1.2	15	342	0	338.0	1.2	15	342	1
rc203-025	326.9	1.0	9	264	0	326.9	1.0	9	264	1
rc204-025	299.7	11.0	28	509	0	299.7	11.0	28	509	1
rc205-025	338.0	0.7	17	147	0	338.0	0.7	17	147	1
rc206-025	324.0	0.7	16	144	0	324.0	0.7	16	144	1
rc207-025	298.3	0.9	12	130	0	298.3	0.9	12	130	1
rc208-025	269.1	1015.8	35	747	0	269.1	1015.8	35	747	1

Table 2: Optimal solutions Series 2: 25 customers

Instance	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r101-050	1044.0	1.3	25	274	2	1044.0	1.3	25	274	1
r102-050	909.0	2.2	26	420	0	909.0	2.2	26	420	1
r103-050	769.7	9.5	60	705	5	772.9	21.1	136	1035	12
r104-050	622.5	100.5	75	1171	15	625.4	319.4	222	3113	20
r105-050	893.7	2.7	37	454	4	899.3	10.7	141	746	20
r106-050	793.0	6.0	37	732	6	793.0	6.0	37	732	1
r107-050	707.6	14.1	52	938	9	711.1	80.5	275	2162	23
r108-050	598.3	253.9	88	1455	0	*687.0*				
r109-050	776.6	5.9	53	670	5	788.6	103.2	476	1814	114
r110-050	696.8	9.2	60	936	2	697.0	12.8	84	1028	3
r111-050	697.7	14.0	62	997	16	707.2	237.5	435	2638	72
r112-050	616.8	31.8	56	1249	8	630.2	3327.3	1581	8584	352
c101-050	362.4	0.9	13	352	0	362.4	0.9	13	352	1
c102-050	361.4	2.5	23	393	0	361.4	2.5	23	393	1
c103-050	361.4	5.2	27	609	0	361.4	5.2	27	609	1
c104-050	358.0	3301.7	36	800	0	358.0	3301.7	36	800	1
c105-050	362.4	1.6	18	363	0	362.4	1.6	18	363	1
c106-050	362.4	1.4	18	317	0	362.4	1.4	18	317	1
c107-050	362.4	1.7	19	408	0	362.4	1.7	19	408	1
c108-050	362.4	1.7	20	216	0	362.4	1.7	20	216	1
c109-050	362.4	3.7	25	393	0	362.4	3.7	25	393	1
rc101-050	944.0	6.8	79	859	31	944.0	6.8	79	859	1
rc102-050	813.8	12.1	91	1133	11	822.5	251.0	845	4658	118
rc103-050	710.9	26.2	113	1619	2	710.9	26.2	113	1619	1
rc104-050	545.8	17.5	33	518	0	545.8	17.5	33	518	1
rc105-050	855.1	9.2	93	999	10	855.3	11.6	126	1261	3
rc106-050	720.0	18.6	89	1040	4	723.2	36.6	185	1757	14
rc107-050	640.1	28.0	103	900	1	642.7	60.9	183	1479	10
rc108-050	596.5	77.5	103	1666	2	598.1	243.8	252	2534	11

Table 3: Optimal solutions Series 1: 50 customers

Instance	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r201-050	791.9	6.9	54	668	0	791.9	7.0	54	668	1
r202-050	698.5	22.9	62	1345	0	698.5	22.9	62	1345	1
r203-050	598.6	125.3	59	1662	0	605.3	381.5	246	3839	9
r204-050	*505.3*					*551.5*				
r205-050	682.9	20.4	70	1224	0	690.1	565.7	759	4840	80
r206-050	626.3	53.7	65	1803	0	*632.4*				
r207-050	564.1	1285.0	79	2713	0	*643.4*				
r208-050	*485.0*					*541.4*				
r209-050	599.8	60.5	74	1165	0	600.6	92.6	133	1472	3
r210-050	636.1	56.9	101	2106	0	*765.6*				
r211-050	528.6	614.9	64	2383	0	*616.2*				
c201-050	360.2	5.9	44	597	0	360.2	5.9	44	597	1
c202-050	360.2	85.8	69	3392	0	360.2	85.8	69	3392	1
c203-050	*359.8*					*359.8*				
c204-050	*350.1*					*350.1*				
c205-050	359.8	14.0	52	1642	0	359.8	14.0	52	1642	1
c206-050	359.8	6.5	28	768	0	359.8	6.6	28	768	1
c207-050	359.6	22.2	22	1241	0	359.6	22.2	22	1241	1
c208-050	350.5	26.5	59	2211	0	350.5	26.6	59	2211	1
rc201-050	684.8	3.4	27	516	0	684.8	3.4	27	516	1
rc202-050	613.6	12.9	44	613	0	613.6	13.0	44	613	1
rc203-050	555.3	702.7	45	1889	0	555.3	702.7	45	1889	1
rc204-050	*448.3*					*448.3*				
rc205-050	630.2	7.7	36	615	0	630.2	7.7	36	615	1
rc206-050	610.0	5.8	27	473	0	610.0	5.8	27	473	1
rc207-050	558.6	46.3	48	892	0	558.6	46.3	48	892	1
rc208-050	*472.9*					*494.0*				

Table 4: Optimal solutions Series 2: 50 customers

Instance	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r101-100	1634.0	14.8	66	1013	7	1637.7	18.3	90	1147	5
r102-100	1466.6	17.5	48	1110	0	1466.6	17.5	48	1110	1
r103-100	1206.8	124.4	111	1965	3	1208.7	265.0	200	2421	8
r104-100	957.3	2531.2	219	3357	0	*1138.7*				
r105-100	1349.3	46.4	115	1679	17	1355.3	260.3	428	3156	38
r106-100	1228.1	123.3	149	2304	13	1234.6	1920.8	771	5469	68
r107-100	1055.5	579.0	186	2874	0	*1402.1*				
r108-100	*915.1*					*1077.0*				
r109-100	1135.1	112.9	143	2136	0	*1356.3*				
r110-100	1056.0	174.7	142	2669	0	*1200.6*				
r111-100	1034.9	195.6	127	2787	0	*1256.5*				
r112-100	927.9	2691.6	162	3087	0	*1060.6*				
c101-100	827.3	7.5	31	566	0	827.3	7.5	31	566	1
c102-100	827.3	16.0	34	1009	0	827.3	16.0	34	1009	1
c103-100	826.3	91.6	55	1569	0	826.3	91.6	55	1569	1
c104-100	*822.9*					*822.9*				
c105-100	827.3	12.2	40	743	0	827.3	12.2	40	743	1
c106-100	827.3	8.4	31	895	0	827.3	8.4	31	895	1
c107-100	827.3	7.9	29	778	0	827.3	7.9	29	778	1
c108-100	827.3	17.6	47	912	0	827.3	17.6	47	912	1
c109-100	827.3	58.1	73	1172	0	827.3	58.1	73	1172	1
rc101-100	1617.4	70.4	161	1907	44	1619.8	222.5	303	2617	16
rc102-100	1441.8	124.3	171	2421	0	*1916.3*				
rc103-100	1245.4	475.8	254	2879	0	*1546.2*				
rc104-100	*1117.6*					*1269.1*				
rc105-100	1509.8	114.2	196	2270	28	1513.7	199.4	317	2658	14
rc106-100	1343.0	213.9	175	2296	0	*1591.2*				
rc107-100	1196.5	449.6	176	2595	0	*1424.0*				
rc108-100	1105.1	3175.5	276	3427	0	*1276.4*				

Table 5: Optimal solutions Series 1: 100 customers

Instance	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r201-100	1140.3	330.6	158	2574	0	1143.2	1104.4	819	5711	44
r202-100	1022.2	1055.9	142	4311	0	*1247.6*				
r203-100	*867.0*					*1027.8*				
r204-100	*743.2*					*852.6*				
r205-100	939.2	848.4	212	3887	0	*1138.9*				
r206-100	*866.9*					*1098.9*				
r207-100	*791.1*					*916.0*				
r208-100	*702.7*					*774.8*				
r209-100	841.4	1777.1	234	4996	0	*1072.6*				
r210-100	*889.4*					*1057.3*				
r211-100	*735.1*					*883.8*				
c201-100	589.1	39.0	78	1572	0	589.1	39.1	78	1572	1
c202-100	589.1	374.2	121	4104	0	589.1	374.3	121	4104	1
c203-100	*588.7*					*588.7*				
c204-100	*598.2*					*715.2*				
c205-100	586.4	360.2	224	4499	0	586.4	360.3	224	4499	1
c206-100	586.0	124.6	97	2923	0	586.0	124.6	97	2923	1
c207-100	585.8	543.0	154	6956	0	585.8	543.2	154	6956	1
c208-100	585.8	261.7	130	4359	0	585.8	261.7	130	4359	1
rc201-100	1255.9	283.7	167	2343	0	1261.8	992.2	858	5204	26
rc202-100	1088.1	949.7	192	3637	0	1092.3	3526.2	804	7318	24
rc203-100	*923.1*					*1138.3*				
rc204-100	*795.8*					*924.9*				
rc205-100	1147.6	869.9	188	2765	0	*1567.8*				
rc206-100	1038.6	2522.5	229	3358	0	*1161.8*				
rc207-100	*947.4*					*1252.4*				
rc208-100	*768.5*					*819.0*				

Table 6: Optimal solutions Series 2: 100 customers

With the time limit set to one hour, 125 instances are solved out of 168: all instances with 25 customers, 45 out of 56 instances with 50 customers, 24 out of 56 with 100 customers. These results compare very favorably with the results reported in the survey of Cordeau *et al.* (2001). In this survey, 126 known optimal solutions are reported, compiled from four different papers, many of which could only be solved with the help of massive computing resources or time (going up to several days); furthermore, 4 of these "optimal" solutions later proved to be wrong. Among the methods presented in the 4 papers considered, which represented the best methods at that time, the method of Kallehauge *et al.* (2001) clearly emerges as the best (Note than an updated version of this paper was later published as Kallehauge *et al.* 2006). Its mean computing time for the instances with 100 customers solved by both our and their algorithms is 2,764 seconds on a HPJ7000, compared to 301 seconds for our algorithm. Also, apart from the 4 wrong instances (that we all solved), our method is able to to solve 10 new instances with a mean computing time of 267 seconds.

A detailed analysis of the results highlights that our method is generally much more effective than other methods for instances with wide time windows, and suffers for some c instances. In both cases, the explanation can be found in the fact that we maintain the elementary path condition in our model. Indeed, the wider the time windows are, the greater the impact of the elementary path condition is. On the contrary, maintaining the elementary path condition does not improve the quality of the linear relaxation for most of the c instances. However, even if optimality is not proved, all but one (c204-100) optimal solutions of the c instances were found.

Beside its efficiency, another interesting feature of our algorithm is its impact on the convergence of the column generation process. In all cases, the objective function decreases quickly toward the optimal value. In difficult instances, the intractability that remains lies in the last iterations of the process. Indeed, except LE, our refinements are not designed to be helpful in proving that no negative reduced cost column exists. Figure 4 illustrates the convergence of the column generation process (at the root node of the search tree) in the case of instances r204-25 and r107-50, where the refinements respectively have strong or weak impacts on computing times. Axes represent time (in seconds) and objective function value. Note that time scales are different with or without refinements.

As mentioned in Section 1.3, since the results reported in Cordeau *et al.* (2001) were obtained, several papers have focussed on the improvement of the lower bounding scheme, enabling a larger set of instances to be solved with reasonable computing times. Chabrier's algorithm (2006) uses the ESPPRC bound and is thus very similar to Feillet *et al.*'s (2004) and to the present algorithm before refinements. This method is able to solve 104 instances in less than an hour, with computing conditions similar to the ones used here.

Irnich and Villeneuve (2003) evaluate the efficiency of forbidding k-cycles in the set of feasible routes. They assess their approach with the elimination of 2-cycles, 3-cycles or 4-cycles respectively. They propose in this way a compromise between the SPPRC and ESPPRC based bounding schemes. Their approach solves respectively 111, 117 and 117 instances for $k = 2$, $k = 3$ and $k = 4$, in less than one hour on a Pentium 600 MHz. It is rather difficult to compare precisely these algorithms with ours, since the machine used is less powerful than ours and since computing times are limited to one hour in their experiments. However, one can notice that most of the instances we solve are solved with a computing time significantly shorter than one hour. Hence, one can reasonably claim that the behavior of their algorithm when 3- or 4-cycles are eliminated is globally comparable to ours. However, it relies on much more complicated algorithmic structures than ours. Furthermore, our refinements are designed to be generic and could certainly be advantageously included in their algorithms or in the application of column generation in other contexts.

Boland *et al.* (2006) and Righini and Salani (2005) propose dynamical solution of the ESPPRC by progressively adding the elementary path constraint, i.e., progressively adding single-visit conditions for the visit of customers. Their approach leads to significant improvements in the efficiency of the dynamic programming algorithm proposed in Feillet *et al.* (2004). However, this approach has not been evaluated within the context of a column generation scheme. Seeing the potential interest of this strategy, we have

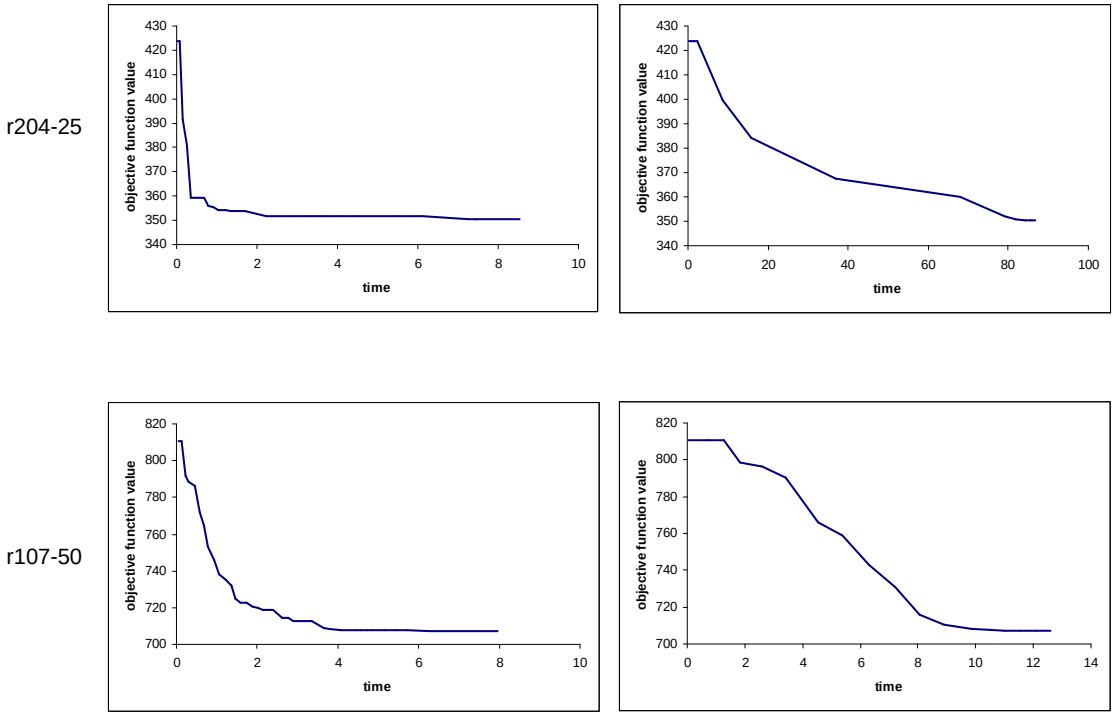

Figure 4: Convergence of column generation for r204-25 and r107-50, with or without refinements.

implemented and evaluated it here. Our implementation works as follows. Dynamic programming begins without including any single-visit condition. Three cases can occur:

- At least one elementary route of negative reduced cost is found; the subproblem stops.
- No elementary route, but at least one non-elementary route of negative reduced cost is found; a single-visit condition is added to the most visited customer (in these routes) and the dynamic programming algorithm is applied again.
- No elementary or non-elementary route of negative reduced cost is found; the DISC parameter (of the LDS policy) is increased; if its value was maximal, the algorithm stops.

Table 7 evaluates the impact of this strategy. It presents the CPU time, the number of subproblem calls and the number of column generated for our algorithm (ESPPRC) and for the variant presented above (Progressive ESPPRC), on the subset of instances solved with a CPU time greater than 100 seconds in Tables 1 to 6.

As can be seen in this table, the results are contrasted. The progressive integration of the elementary path condition does not induce a clear improvement of computing times. This seems related to the sometimes large increase in the number of subproblems solved. Indeed, many iterations stop with a very limited number of new columns found. One might need a more advanced integration of this new strategy with the existing heuristic features of the subproblem solution (LDS, premature stopping, etc.) to obtain a clear positive impact.

Finally, a very recent and efficient implementation of Branch and Price is proposed by Desaulniers *et al.* (2006). The resulting algorithm is rather complex, but provides the most efficient approach up to now.

	Progressive ESPPRC			ESPPRC		
Instance	CPU	Iter	Col	CPU	Iter	Col
c204-025	464.3	74	765	540.2	48	1013
rc208-025	1855.9	55	511	1015.8	35	747
r104-050	128.9	247	1739	319.4	222	3113
r109-050	100.3	550	1945	103.2	476	1814
r111-050	136.4	426	2091	237.5	435	2638
r112-050	4826.9	2598	8304	3327.3	1581	8584
c104-050	512.5	92	543	3301.7	36	800
rc102-050	258.8	935	4160	251.0	845	4658
rc108-050	317.6	457	2390	243.8	252	2534
r203-050	205.9	485	2644	381.5	246	3839
r205-050	617.9	1577	5114	565.7	759	4840
rc203-050	419.7	94	895	702.7	45	1889
r103-100	172.7	297	2429	265.0	200	2421
r105-100	230.8	496	3225	260.3	428	3156
r106-100	1096.2	873	4691	1920.8	771	5469
rc101-100	183.6	276	2307	222.5	303	2617
rc105-100	123.4	293	2274	199.4	317	2658
r201-100	1093.6	1319	6038	1104.4	819	5711
c202-100	107.0	115	2453	374.3	121	4104
c205-100	172.7	274	2830	360.3	224	4499
c206-100	267.0	255	3433	124.6	97	2923
c207-100	436.3	304	3856	543.2	154	6956
c208-100	201.6	201	2392	261.7	130	4359
rc201-100	1528.5	2088	7483	992.2	858	5204
rc202-100	3182.7	1893	8135	3526.2	804	7318

Table 7: Impact of the progressive introduction of elementary path constraints

It clearly outperforms our method and solves most instances, often with quite short computing times. Apparently, the main reasons of this success lie in:

- the use of a tabu search heuristic to find new columns,
- the efficient generation of cuts (notably the subset row inequalities proposed by Jepsen *et al.* (2006)),
- the use of a new lower bounding scheme, inspired by Boland *et al.* (2006) and Righini and Salani (2005), authorizing multiple visits only for a subset of customers (dynamically computed).

3.2 Detailed Analysis of the Impact of Refinements

In this subsection, we propose some complementary numerical results, evaluating further the impact of the refinements. These results are presented in Table 8 and Table 9. For this purpose, a limited set of instances is used: the ones for which the LP relaxation is found between 10 and 100 seconds in Table 1 to Table 6. This criterion leads to the selection of 32 instances with diverse characteristics. The LP relaxation is solved again for these instances using several versions of the algorithm. Since we focus on the refinements, in these tests we include neither the computation of 2-path cuts nor the tree search. We evaluate five solution schemes: a scheme without refinement, a scheme with all the refinements and three schemes including respectively each one of the three refinements alone. In each case, the tables present the CPU time needed for solving the linear relaxation and the total number of iterations and columns generated.

Some obvious conclusions can be drawn from these tables. First, the impact of the refinements is clarified in Table 8. In addition to the impressive speeding up of the solution process for difficult instances, the refinements have a significant impact on the number of columns generated, which is much smaller, but many more iterations are often needed.

It is important to note that the impact of the refinements is rather negligible, and in some cases even detrimental, for instances that are solved very quickly (e.g., rc106-050). This is due to the fact that the refinements imply some additional work that is not offset by the improvements when an instance is easy. In general, the more difficult the instances are to solve, the greater is the impact of the refinements. The speedup can even be as high as two orders of magnitude (e.g., c207-050). Out of the 32 instances of this set, 9 display speedups that are larger than an order of magnitude and 7 others speedups in the range 3-10. Overall, the total time required to solve the relaxation of these 32 instances is cut down by a factor of 18.6 when using the refinements and every instance can be solved in 90.9 seconds or less.

Table 9 exhibits clearly the efficiency of Label Loading and Meta Extension. These combined techniques achieve a drastic improvement in terms of computing time and set of columns generated. However, their impact on the number of iterations is very limited.

The impact of LDS is less consistent. It sometimes gives excellent results (e.g., instance c103-100), but in other cases it has a slight negative effect (e.g., r206-50). In most cases, it replaces some slow iterations with several quick ones and produces a smaller set of generated columns. Also, and that is the most interesting point, it is very complementary with LL and ME, and the best results are obtained when all three techniques are combined.

Finally, Label Elimination does not have a significant impact on the efficiency of the solution scheme. Further results show that, for the selected instances, about 25% of the labels are eliminated (with a very large deviation, the reduction approximately going from 0% to 70%); however, the total number of labels processed is only reduced by 0.3% on average when LE is used. This small reduction indicates that, in most cases, the labels eliminated would not have been extended, had they been kept. Figure 5 illustrates this behavior and also shows that no clear relationship exists between the reduction in number of labels processed and the reduction in computing times. After seeing the inefficiency of LE, we tested our algorithm with all refinements but LE. Removing LE did not induce significant changes to the results reported previously.

	without refinements			with all refinements		
Instance	CPU	Iter	Col	CPU	Iter	Col
r204-025	84.2	12	2136	7.0	28	772
r208-025	396.8	16	3352	18.2	24	1183
rc204-025	676.8	8	1099	11.0	28	509
r107-050	12.5	17	1551	9.4	42	853
r111-050	11.3	17	1355	7.8	47	923
r112-050	39.7	20	1998	11.0	41	1205
rc102-050	4.5	14	1089	4.6	35	740
rc103-050	19.6	16	1204	10.8	41	861
rc104-050	151.0	15	1077	18.1	33	518
rc106-050	3.7	14	993	6.1	42	753
rc107-050	22.7	15	1293	14.4	63	563
rc108-050	109.8	15	1418	13.6	36	857
r202-050	31.6	20	3455	22.3	62	1345
r205-050	17.8	23	2746	14.1	70	1224
r206-050	248.7	25	5318	45.1	65	1803
r209-050	139.6	25	3974	30.9	74	1165
r210-050	205.3	25	5106	54.9	101	2106
c202-050	2418.5	69	15966	86.2	69	3392
c205-050	889.8	123	15366	13.4	52	1642
c207-050	2269.5	61	15281	22.3	22	1241
c208-050	1943.6	95	23237	29.8	59	2211
rc202-050	33.2	17	1433	13.1	44	613
rc207-050	181.5	20	2616	44.3	48	892
r101-100	7.2	19	1162	8.8	35	869
r102-100	48.3	25	2398	18.0	48	1110
r105-100	28.7	29	2304	30.8	82	1522
c102-100	546.9	24	2354	18.2	34	1009
c103-100	3933.6	38	4635	90.9	55	1569
c105-100	20.6	25	1008	12.3	40	743
c108-100	52.9	28	1252	17.7	47	912
c109-100	135.5	28	1780	58.1	73	1172
rc101-100	20.1	30	2170	27.8	87	1400
Mean value	459.5	29.0	4128.9	24.7	50.8	1177.4

Table 8: Global impact of the refinements

	with LDS			with LL and ME			with LE		
Instance	CPU	Iter	Col	CPU	Iter	Col	CPU	Iter	Col
r204-025	22.8	43	1430	11.4	12	1004	78.5	12	2136
r208-025	45.5	46	2642	49.9	17	1566	354.9	16	3352
rc204-025	11.4	26	522	66.3	6	643	665.5	8	1099
r107-050	12.3	60	1236	10.2	22	1107	12.3	17	1551
r111-050	13.3	76	1228	10.4	21	1012	11.4	17	1355
r112-050	17.9	66	1495	33.1	27	1486	40.4	20	1998
rc102-050	7.4	61	1160	5.9	23	971	3.9	14	1089
rc103-050	13.7	60	1265	18.5	24	1119	16.0	16	1204
rc104-050	38.1	37	659	136.9	17	649	83.0	15	1077
rc106-050	9.0	67	1075	3.3	16	710	3.8	14	993
rc107-050	24.3	71	1092	24.4	22	919	17.9	15	1293
rc108-050	15.7	42	838	80.0	17	897	84.1	15	1418
r202-050	40.9	81	2455	13.3	20	1427	31.5	20	3455
r205-050	49.4	140	2436	9.7	25	1228	18.7	23	2746
r206-050	269.6	154	4305	45.7	29	1734	253.1	25	5318
r209-050	105.3	148	2972	35.6	29	1397	140.8	25	3974
r210-050	223.3	144	3812	109.4	41	2354	210.3	25	5106
c202-050	734.7	219	11617	116.2	42	3639	2378	69	15966
c205-050	786.0	294	11229	9.2	20	1050	894.8	123	15366
c207-050	763.7	219	11147	155.0	38	2617	2293.0	61	15281
c208-050	1030.4	223	11172	53.2	29	1734	2007.1	95	23237
rc202-050	33.0	78	1776	17.1	20	1051	34.1	17	1433
rc207-050	95.5	72	1319	133.9	25	1848	197.3	20	2616
r101-100	23.3	91	1607	8.0	22	972	7.2	19	1162
r102-100	58.7	104	2227	34.4	28	1538	41.3	25	2398
r105-100	64.7	150	3255	28.5	44	1671	23.8	29	2304
c102-100	60.3	109	2085	107.0	28	1388	485.5	24	2354
c103-100	184.8	75	2087	1111.7	33	2293	3066.2	38	4635
c105-100	40.5	111	1306	20.4	27	622	18.4	25	1008
c108-100	36.2	67	1287	34.0	24	915	48.0	28	1252
c109-100	89.9	100	1710	112.0	33	1276	128.9	28	1780
rc101-100	51.7	141	2916	23.6	46	1687	18.8	30	2170
Mean value	155.4	105.5	3042.6	82.1	25.8	1391.4	427.1	29.0	4128.9

Table 9: Impact of the different refinements

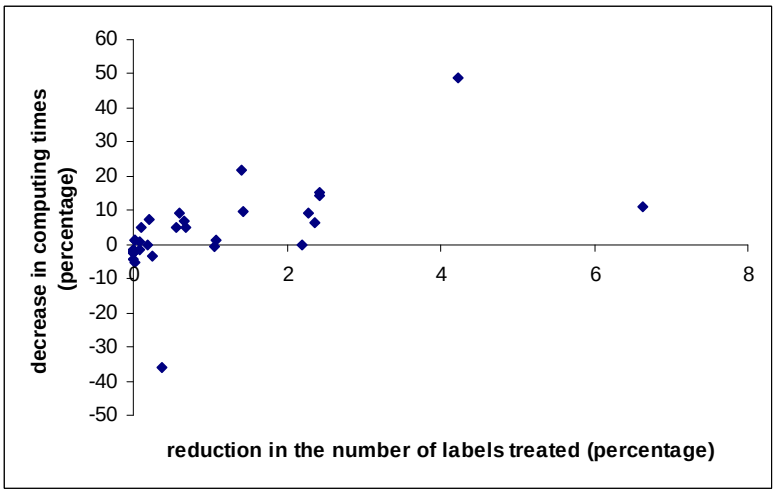

Figure 5: Relationship between the reductions in the number of labels processed and computing times.

4 Conclusion

In this paper, we have presented three techniques that accelerate column generation schemes for vehicle routing problems. We have applied these techniques to the VRPTW, where the computational experiments have demonstrated their important impact on the efficiency of the solution scheme, especially when dealing with difficult instances. Limited Discrepancy Search allows to rapidly execute the first iterations of column generation and to concentrate the effort on the last iterations. Label Loading and Meta Extension are simple techniques that prove very efficient, notably because they transform the traditional label extension procedure into more powerful local search operators. Finally, by computing a lower bound after each extension, we are able to identify and remove a large number of labels that can be shown to be worthless. This last technique, which can be seen as an implementation of the ideas independently sketched in Lübbecke (2005), proved rather inefficient in our case.

All these techniques together, but especially LDS, Label Loading and Meta Extension, have enabled us to solve quickly many instances that could only be solved before with unduly long computing times. Furthermore, these techniques provide generic tools to accelerate the convergence of column generation approaches: good columns are found quickly, which leads to fast convergence toward the optimal solution. The main difficulties then remain in the last iterations where interesting columns can be very difficult to find. Thereby, some issues, such as the generation of a good initial set of columns or the use of heuristic algorithms for trying to generate columns first, tend to become irrelevant.

Very recently, other attempts for solving efficiently the VRPTW using column generation have emerged, sometimes with a greater success than ours (Desaulniers *et al.* 2006). However, in our opinion, the main contributions of our refinements relate to their simplicity and their generic nature. The LDS principle can easily be applied to any subproblem solution algorithm as soon as an enumerative scheme (Dynamic Programming, Branch and Bound, Constraint Programming) is applied, in the context of vehicle routing or not; developing an ad hoc metaheuristic, though possibly more efficient, is certainly much longer and more complicated. The basic idea of Label Loading and Meta Extension should also be very easy to adapt to many situations. Furthermore, it is quite original compared to other approaches proposed in the litterature

and could certainly be included advantageously in the most efficient methods cited above.

Another interesting point in our approach is its robustness against difficult problems where imposing the elementary path condition is crucial to maintain the quality of the lower bound. This is the case of several problems like the Team Orienteering Problem, the Capacitated Team Orienteering Problem or the Capacitated Profitable Tour Problem. Our refinements were all easily applied with success on these problems (Boussier *et al.* forthcoming, Archetti *et al.* 2007).

References

[1] Archetti C., Feillet D., Hertz A., and Speranza M.G. The capacitated team orienteering and profitable tour problems. Technical Report 2007-943, LIA,Université d'Avignon, France, 2007.

[2] Boland N., Dethridge J., and Dumitrescu I. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34:58–68, 2006.

[3] Boussier S., Feillet D., and Gendreau M. An exact algorithm for team orienteering problems. *4OR*, forthcoming.

[4] Chabrier A. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33:2972–2990, 2006.

[5] Cook W. and Rich J.L. A parallel cutting-plane algorithm for the vehicle routing problem with time windows. Technical Report TR99-04, Department of Computational and Applied Mathematics,Rice University, 1999.

[6] Cordeau, J.F., Desaulniers G., Desrosiers J., Solomon M.M., and Soumis F. The VRP with time windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, pages 157–194. 2001.

[7] Desaulniers G., Desrosiers J., and Solomon M.M. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in Metaheuristics*, pages 309–324. Kluwer, 2001.

[8] Desaulniers G., Lessard F., and Hadjar A. Tabu search, generalized k-path inequalities, and partial elementarity for the vehicle routing problem with time windows. Technical Report G-2006-45, GERAD,Canada, 2006.

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

[10] Desrosiers J., Dumas Y., Solomon M.M., and Soumis F. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monna, and G.I. Nemhauser, editors, *Network Routing*, Handbooks in Operations Research and Management Science, pages 35–139. Amsterdam, North-Holland, 1995.

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

[12] Gelinas S., Desrochers M., Desrosiers J., and Solomon M.M. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61:91–109, 1995.

[13] Harvey W. and Ginsberg M. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 607–615, Montréal, Canada, 1995. Morgan Kaufmann.

[14] Irnich S. The shortest path problem with k-cycle elimination (k≥3): Improving a branch and price algorithm for the VRPTW. In *TRISTAN IV*, volume 3, pages 571–574, 2001.

[15] Irnich S. and Villeneuve D. The shortest path problem with k-cycle elimination (k≥3): Improving a branch and price algorithm for the VRPTW. Technical Report G-2003-55, GERAD, 2003.

[16] Jepsen M., Petersen B., Spoorendonk S., and Pisinger D. A non-robust branch-and-cut-and-price algorithm for the vehicle routing problem with time windows. Technical Report 06-03, Department of Computer Science, University of Copenhagen, Denmark, 2006.

[17] Kallehauge B., Larsen J., and Madsen O.B.G. Lagrangean duality applied on vehicle routing with time windows - experimental results. Technical Report IMM-TR-2001-9, IMM, Technical University of Denmark, 2001.

[18] Kallehauge B., Larsen J., and Madsen O.B.G. Lagrangean duality applied to the vehicle routing problem with time windows. *Computers & Operations Research*, 33(5):1464–1487, 2006.

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

[20] Lübbecke M.E. Dual variable based fathoming in dynamic programs for column generation. *European Journal of Operational Research*, 162(1):122–125, 2005.

[21] Righini G. and Salani M. New dynamic programming algorithms for the resource-constrained elementary shortest path problem. Technical Report 66, University of Milano, Italy, 2005.

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

[23] Rousseau L.-M., Gendreau M., and Feillet D. Interior point stabilization for column generation. *Operations Research Letters*, 35:561–592, 2007.

[24] Sol M. *Column generation techniques for pickup and delivery problems.* PhD thesis, Technische Universiteit Eindhoven, Netherlands, 1994.

[25] Solomon M.M. *Vehicle Routing and Scheduling with Time Window Constraints: Models and Algorithms.* PhD thesis, Department of Decision Sciences, University of Pennsylvania, 1983.