A Branch-and-Price-Based Large Neighborhood Search Algorithm for the Vehicle Routing Problem with Time Windows

Eric Prescott-Gagnon and Guy Desaulniers
École Polytechnique de Montréal and GERAD, C.P. 6079, Succursale Centre-Ville, Montréal, Québec, Canada H3C 3AT

Louis-Martin Rousseau
École Polytechnique de Montréal and CIRRELT, C.P. 6079, Succursale Centre-Ville, Montréal, Québec, Canada H3C 3AT

Given a fleet of vehicles assigned to a single depot, the vehicle routing problem with time windows (VRPTW) consists of determining a set of feasible vehicle routes to deliver goods to a set of customers while minimizing, first, the number of vehicles used and, second, total distance traveled. A large number of heuristic approaches for the VRPTW have been proposed in the literature. In this article, we present a large neighborhood search algorithm that takes advantage of the power of branch-and-price which is the leading methodology for the exact solution of the VRPTW. To ensure diversification during the search, this approach uses different procedures for defining the neighborhood explored at each iteration. Computational results on the Solomon's and the Gehring and Homberger's benchmark instances are reported. Compared to the best known methods, the proposed algorithm produces better solutions, especially on the largest instances where the number of vehicles used is significantly reduced. © 2009 Wiley Periodicals, Inc. NETWORKS, Vol. 54(4), 190–204 2009

Keywords: vehicle routing; time windows; large neighborhood search; heuristic column generation

1. INTRODUCTION

Given a fleet of vehicles assigned to a single depot, the vehicle routing problem with time windows (VRPTW) consists of determining a set of feasible routes (one route per vehicle used) to deliver goods to a set $\mathcal{N}$ of scattered customers while minimizing, first, the number of vehicles used and, second, the total distance traveled (which is usually proportional to the total traveling cost). Each customer $i \in \mathcal{N}$ must be visited exactly once by one vehicle within a prescribed time interval $[a_i, b_i]$, called a time window, to deliver a quantity q_i of goods. A route starts from the depot and visits a sequence of customers before returning to the depot. It is feasible if the total amount of goods delivered does not exceed the vehicle capacity Q and if it respects the time window of each visited customer.

The VRPTW can be represented on a graph $G = (V, \mathcal{A})$. The node set V contains $|\mathcal{N}| + 2$ nodes: one node for each customer $i \in \mathcal{N}$, as well as one source node o and one sink node d representing the depot at the beginning and the end of the planning horizon, respectively. The arc set $\mathcal{A}$ contains start arcs (o, j), $\forall\, j \in \mathcal{N}$, end arcs (i, d), $\forall\, i \in \mathcal{N}$, and travel arcs (i, j), $\forall\, i, j \in \mathcal{N}$ such that customer j can be visited immediately after customer i in at least one feasible route, that is, if $a_i + t_{ij} \leq b_j$, where t_{ij} is equal to the travel time between i and j plus the service time (if any) at node i. Each arc (i, j) has an associated travel cost (distance) c_{ij}. Note that, in general, c_{ij} is proportional to the travel time or the distance between i and j.

The VRPTW has been well studied in the literature. In the past 15 years, several exact methods for the VRPTW have been developed. Among them, branch-and-price algorithms [6, 10, 11, 13, 22–24] have produced the best results, mostly because of the quality of the lower bounds yielded by the embedded column generation method. Also, a very large number of heuristics have been proposed (see the survey papers of [4, 5]), including a wide variety of metaheuristics such as tabu search [7], evolutionary and genetic algorithms [2, 26], large neighborhood search [27, 29], variable neighborhood search [3, 28], certain two-phase hybrid approaches [1, 15, 18], iterated local search algorithms [19, 20], and one parallel cooperative search approach that exploits several known metaheuristics [25]. However, to our knowledge, no heuristics taking advantage of the power of branch-and-price have been proposed in the literature for the VRPTW.

Received September 2007; accepted March 2008
Correspondence to: G. Desaulniers; e-mail: guy.desaulniers@gerad.ca
DOI 10.1002/net.20332
Published online 11 August 2009 in Wiley InterScience (www.interscience.wiley.com).

In this article, we present such a method, namely a large neighborhood search (LNS) algorithm, that relies on a heuristic branch-and-price method for neighborhood exploration. An LNS method is an iterative method where elements of a solution are alternately removed (destruction step) and reinserted (reconstruction step) in order to improve the solution. A neighborhood is thus the set of all solutions containing the subset of elements that have not been removed at a given iteration. Because the size of the neighborhood increases exponentionally with the number of elements removed, it has the potential to change a large portion of the solution, hence its name. Like the approaches of Gehring and Homberger [15], Bent and Van Hentenryck [1], and Homberger and Gehring [18], our method has two phases: in the first, the minimization of the number of vehicles is prioritized; in the second, the priority is changed to reducing total traveled distance with a fixed number of vehicles, namely, the minimum number attained in the first phase. However, as opposed to these three hybrid approaches, our method applies a very similar methodology in both phases.

To evaluate the efficiency of the proposed LNS method, we performed computational experiments on the well-known 56 Solomon's [30] instances with 100 customers and also on the 300 Gehring and Homberger's [14] instances involving between 200 and 1,000 customers. Compared to the best known methods, our LNS method produced better solutions, especially on the largest instances where the number of vehicles used was significantly reduced. It succeded to compute 145 new best solutions out of the 356 benchmark instances. However, it required more computational time than other leading methods such as that of Pisinger and Ropke [27].

Combining mathematical programming techniques with metaheuristics is a growing area of research. For instance, De Franceschi et al. [8] developed an LNS algorithm for the distance-constrained vehicle routing problem (without time windows) where the reconstruction step consists of solving a set partitioning type problem using a commercial mixed-integer programming solver. Their method differs from the one we propose in several ways. Firstly, we use different ad hoc operators to destroy the current solution at each LNS iteration. Secondly, reconstruction is performed using a branch-and-price heuristic. Finally, the first phase of our two-phase method aims at minimizing the number of vehicles used, while De Franceschi et al. [8] consider a fixed number of vehicles. Our work thus brings an interesting contribution in this research trend.

The article is organized as follows. In the next section, we present the LNS algorithm framework, with an emphasis on the description of the two-phase solution process. In Section 3, we present the set of destruction operators that can be used to define the neighborhood to explore at each iteration of the LNS algorithm. Most of these operators are adaptations of operators already proposed in the literature. Section 4 describes the branch-and-price heuristic applied in the reconstruction step at each LNS iteration. This heuristic is composed of a heuristic branching strategy and a heuristic column generation method where columns (routes) are generated by tabu search. The results of our computational experiments are reported in Section 5. Finally, conclusions are drawn in Section 6.

2. ALGORITHM FRAMEWORK

An LNS algorithm is an iterative process that destroys at each iteration a part of the current solution using a chosen neighborhood definition procedure and reconstructs it in the hope of finding a better solution. Figure 1 illustrates the framework of our LNS algorithm that works in a two-phase fashion. The first one, the vehicle number reduction phase (VNR), as its name indicates, tries to reduce the total number of vehicles used in the current solution, while the other one, the total distance reduction phase (TDR) tries to reduce the total mileage for a fixed number of vehicles. In this figure, the numbers in circles indicate the algorithm step numbers. Note that the two main steps (destruction and reconstruction) of an LNS algorithm appear in double-lined rectangles (Steps 8 and 9). The details of these two steps will be given in the following sections.

The algorithm starts in Step 1 by computing an initial solution using Solomon's [30] I1 insertion heuristic. In Step 2, it computes a lower bound m_{lb} on the optimal number of vehicles using the vehicle capacity Q and the total customer demand:

$$m_{\text{lb}} = \left\lceil \frac{\sum_{i \in \mathcal{N}} q_i}{Q} \right\rceil. \tag{1}$$

It also sets an upper bound m_{ub} on this number to the number of vehicles used in the initial solution, an iteration counter i to 0, and the current phase at VNR (Step 3).

The VNR phase is skipped if $m_{\text{ub}} = m_{\text{lb}}$ (Step 6). Otherwise, it begins by lowering the upper bound m_{ub} by one (Step 7). This upper bound is enforced at each iteration during reconstruction, while allowing (with a penalty cost) some customers not to be serviced. If no feasible solution (covering all customers) can be found within a prespecified number of iterations $I_{\text{VNR}}^{\max}$ (Step 12), the search is abandoned for that bound and the TDR phase is started from the best feasible solution found so far. Otherwise, the upper bound m_{ub} is lowered again by one (Step 7) and the algorithm has another $I_{\text{VNR}}^{\max}$ iterations to find a feasible solution and so on. Note that, whenever a feasible solution with m_{lb} vehicles is obtained (case yes in Step 6), the VNR phase is stopped and the TDR phase starts from that last solution. In the TDR phase, $I_{\text{TDR}}^{\max}$ iterations are performed (tested in Step 15).

At each iteration of the LNS algorithm (either in the VNR or the TDR phase), a neighborhood-defining operator is selected (Step 8) within a set of four different operators using a roulette-wheel that favors operators which were the most successful at improving the current solution in the past iterations. These four operators are described in Sections 3.1–3.4, while the roulette-wheel procedure is discussed in Section 3.5. The selected operator is then applied to destroy parts of the current solution. The destruction consists of determining a subset of

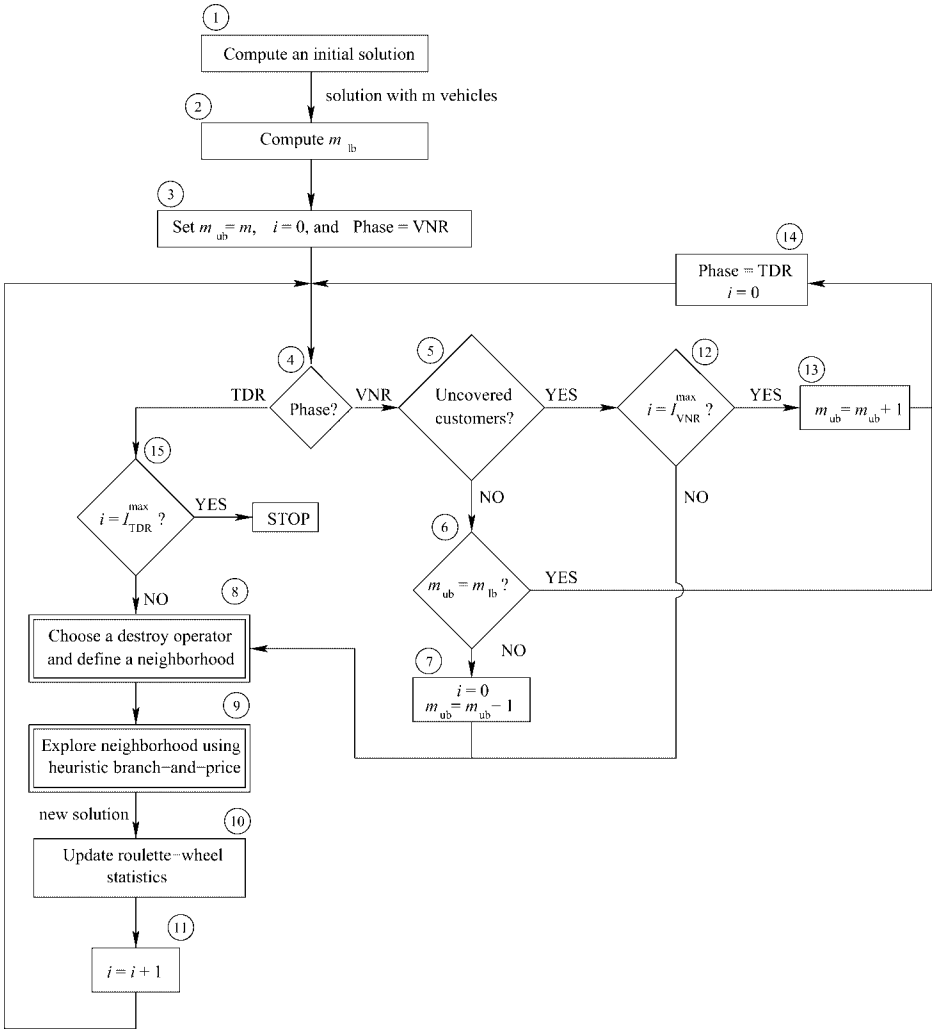

FIG. 1. Algorithm framework.

customers (hereafter called the removed customers) which are disconnected from their current routes, leaving partial routes and isolated customers. The neighborhood contains all solutions that respect the partial routes. These solutions must be feasible except, during the VNR phase, where they might not cover all customers. Reconstruction is performed afterwards in Step 9 by solving the resulting reoptimization problem that corresponds to the original VRPTW where parts of the current solution routes are fixed. This restricted VRPTW is solved using a branch-and-price heuristic to create a new solution. This heuristic is described in Section 4.1. Roulette-wheel statistics are then updated (Step 10) and the process starts over again.

3. DESTRUCTION

To diversify the search, we consider four different neighborhood operators that can be used to destroy the current solution in Step 8 of the algorithm. Each operator tries to select a predetermined number of customers that are to be removed from the current solution. In the VNR phase, uncovered customers might not to be selected except when there are less than ten of them, in which case they are all removed from the solution. When a covered customer is removed, adjacent arcs are removed from the solution, leaving partial routes. To choose at each iteration which operator to use, a roulette-wheel procedure is invoked. Each operator and the roulette-wheel are slightly modified in the VNR phase to promote the insertion of uncovered customers into routes. Note that the part of the solution that has not been destroyed is fixed in the reconstruction process, including the uncovered customers that have not been selected (they will remain uncovered).

3.1. Proximity Operator

The proximity operator is an extension of an operator proposed by Shaw [29]. Its goal is to select customers that are related geographically and temporally. An initial customer is chosen randomly and added to an empty pool of removed customers. For each additional customer, the selection procedure has three steps as follows. First, a seed customer i is chosen randomly in the pool of removed customers. Second, all remaining customers are ranked in decreasing order of

a spatio-temporal proximity measure to i. This measure is defined by

$$R(i,j) = \frac{1}{\left(\frac{\min\{c_{ij},c_{ji}\}}{c_i^{\max}} + \frac{1}{T_{ij}+T_{ji}}\right)}, \quad (2)$$

where c_{ij} is the cost of the arc from i to j and $c_i^{\max}$ is the largest arc cost $c_{i\ell}$ or $c_{\ell i}$ for all customers ℓ that have not been removed yet. Similar to a time window proximity measure defined by Gendreau et al. [16], T_{ij} is equal to $\max\{1, \min\{b_j, b_i+t_{ij}\} - \max\{a_j, a_i+t_{ij}\}\}$. In this expression, $\min\{b_j, b_i+t_{ij}\} - \max\{a_j, a_i+t_{ij}\}$ is the width of the interval of the feasible visiting times at customer j when customer i is visited immediately before j, if possible. If an arc exists from i to j but not from j to i, we set $c_{ji} = \infty$, and, if no arc exists either way, $R(i,j)$ takes value 1. In the third step, the next customer to remove is chosen randomly among the remaining customers, favoring those having a greater proximity measure to i. Indeed, the rank of the customer to be chosen is $\lceil N\rho^D \rceil$, where N is the number of remaining customers, ρ a number generated randomly between 0 and 1, and D a constant greater or equal to 1. A D value of 1 results in complete randomness while an infinite value yields complete determinism. For our experiments, the value of D was set at 35.

In the VNR phase, the seed customer is chosen randomly among the uncovered customers. The other uncovered customers are removed only if they are chosen according to their spatio-temporal proximity.

3.2. Route Portion Operator

The proximity operator described above can remove single customers on certain routes, leaving almost no flexibility to change these routes. To avoid this drawback, the route portion operator, presented as the SMART (SMAll RouTing) operator by Rousseau et al. [28], removes portions of routes around pivot customers. The first pivot is chosen randomly, then adjacent customers on the same route of the current solution are removed as well. A second pivot customer on another route is chosen based on the spatio-temporal proximity measure to the first pivot (defined in the preceding section). Adjacent customers are removed and a third pivot is chosen based on the proximity measure to the first pivot as well, and so on until enough customers are removed or all routes are covered by a pivot since there can only be one pivot per route.

Adjacent customers are chosen according to a maximum distance to the pivot customer (distance is measured along the route, not directly between the customers). All customers on each side of the pivot within the maximum distance as well as the first customer outside of this distance are removed. The maximum distance $\bar{d}$ is set once per call to this operator as follows. Let j be the first pivot customer, and i and k its immediate preceding and succeding customers (or depot) on its route, respectively. Then, $\bar{d} = f_{\text{rp}}\ \max\{c_{ij}, c_{jk}\}$, where the multiplicator f_{rp} evolves throughout the solution process. The first time the route portion operator is called, f_{rp} has a value of 1. If less than the target number of customers have been chosen after selecting a pivot in each route, f_{rp} is multiplied by this target number divided by the number of actually removed customers for the next call to the operator. This ensures that, after a few calls to this operator, the right number of customers can be removed from solutions having fewer routes. Furthermore, f_{rp} also adapts to the fact that the distance between adjacent customers typically lowers as the solution process evolves.

Like for the proximity operator, the first pivot in the VNR phase is chosen randomly between the uncovered customers and, in this case, the maximum distance is equal to the distance from this pivot to the depot. Other uncovered customers can be chosen as subsequent pivots. When an uncovered customer is selected as a pivot, no adjacent customers are removed.

3.3. Longest Detour Operator

The longest detour operator was presented as well by Rousseau et al. [28]. The purpose of this operator is to remove the customers yielding in the current solution the largest distance increases for servicing them. All the customers are first ordered decreasingly by the detour they generate in the current solution. For a customer j being serviced between i and k (customers or depot), its associated detour is equal to $c_{ij} + c_{jk} - c_{ik}$. Like for the proximity operator, the customers are removed randomly, favoring those generating a greater detour.

In the VNR phase, uncovered customers do not generate any detour. Therefore, to also select uncovered customers in this phase, each time that a covered customer i is removed according to its detour, the uncovered customer which is the closest to customer i according to the spatio-temporal proximity measure is also removed unless it was previously removed. In this latter case, no other uncovered customer is selected with respect to i.

3.4. Time Operator

The time operator simply removes customers that are serviced almost at the same time. It first selects randomly a specific time t_s within the horizon. The customers are then ordered increasingly according to a value v_i defined for each customer i as

$$v_i = \begin{cases} a'_i - t_s & \text{if } t_s < a'_i \\ 0 & \text{if } a'_i \le t_s \le b'_i \\ t_s - b'_i & \text{if } t_s > b'_i \end{cases} \quad (3)$$

where $[a'_i, b'_i]$ is the interval of times at which customer i can be visited along its route in the current solution while it remains feasible. The bounds of this interval are computed recursively along this route as follows:

$$a'_{i(0)} = a_{\min} \quad (4)$$

$$a'_{i(r)} = \max\left\{a_{i(r)}, a'_{i(r-1)} + t_{i(r-1),i(r)}\right\} \quad r = 1, \ldots, r_{\max} \quad (5)$$

$$b'_{i(r_{\max})} = b_{\max} \tag{6}$$

$$b'_{i(r)} = \min\left\{b_{i(r)}, b'_{i(r+1)} - t_{i(r),i(r+1)}\right\} \qquad r = r_{\max} - 1, \ldots, 0 \tag{7}$$

where r is the rank of the customer on the route, ranks 0 and $r_{\max}$ correspond both to the depot, and $i(r)$ is the customer (or depot) in rank r. $a_{\min}$ and $b_{\max}$ are the earliest departure time from the depot and the latest arrival time at the depot, respectively. Once the customers are ordered, they are chosen with the same procedure presented for the proximity operator.

In the VNR phase, the time interval of an uncovered customer i is its time window $[a_i, b_i]$.

3.5. Roulette-Wheel Procedure

At each iteration of the LNS algorithm, the roulette-wheel procedure selects in Step 8 which operator is applied to define the neighborhood. It is similar to the procedure introduced in Pisinger and Ropke, 2007 [27]. Each destruction operator is assigned a value π_i that starts at 5. Each time that an operator i is selected and allows to improve the current solution, π_i is incremented by 1. At the beginning of each LNS iteration, the destruction operator is selected randomly, where each operator i has a probability of $\frac{\pi_i}{\sum_i \pi_i}$ of being selected.

In the VNR phase, the π_i value of an operator that yields a solution with fewer uncovered customers is increased by 2 in order to prioritize operators which are best at reducing the number of uncovered customers.

4. RECONSTRUCTION

At each iteration of the LNS algorithm, the VRPTW restricted to the selected neighborhood can be modeled as a set partitioning problem where the variables are feasible routes. Let Ω be a subset of all feasible routes (respecting the fixed parts of the solution). With each route $p \in \Omega$, associate the following parameters: c_p, its cost and $v_{ip}, \forall i \in \mathcal{N}$, taking value 1 if customer i is serviced by route p and 0 otherwise. Finally, a binary variable θ_p is defined for each route $p \in \Omega$, taking a value of 1 if route p is part of the solution and 0 otherwise. With this notation, the restricted VRPTW can be formulated as:

$$\text{Minimize} \quad \sum_{p\in\Omega} c_p\theta_p \tag{8}$$

$$\text{subject to:} \quad \sum_{p\in\Omega} v_{ip}\theta_p = 1, \quad \forall\, i \in \mathcal{N} \tag{9}$$

$$\sum_{p\in\Omega} \theta_p \le m_{ub}, \tag{10}$$

$$\theta_p \text{ binary}, \quad \forall\, p \in \Omega. \tag{11}$$

The objective function (8) aims at minimizing total cost. Set partitioning constraints (9) ensure that each customer is visited exactly once by one vehicle. Constraint (10) imposes an upper bound on the number of vehicles that can be used. Finally, (11) provide binary requirements on the variables.

In general, an LNS algorithm must perform many iterations to reach very good quality solutions. Therefore, to keep computational times relatively low, the reconstruction process must be fast and effective. We thus propose to solve, in Step 9 of the overall algorithm, model (8)–(11) using a heuristic branch-and-price method, that is, a heuristic column generation method embedded into a heuristic branch-and-bound search. The column generation method and the branching scheme are described in the following sections.

4.1. Heuristic Column Generation

Column generation is used to solve the linear relaxation of model (8)–(11). Column generation is an iterative process that solves at each iteration this linear relaxation restricted to a subset of the variables. This restricted problem is called the restricted master problem (RMP). The dual solution of the RMP is then transfered to a subproblem whose objective is to generate negative reduced cost variables to be added back to the RMP. The latter is then solved again with the augmented subset of variables. An exact column generation method ends when the optimal solution of the subproblem has a nonnegative reduced cost. We can conclude thereof that the solution to the RMP is optimal for the whole linear relaxation because no more negative reduced cost variables exist.

The RMP is solved by a linear programming algorithm such as the primal simplex method. For the VRPTW, the column generation subproblem is an elementary shortest path problem with resource constraints (ESPPRC) defined on a restricted network which guarantees that the fixed parts of the routes in the current solution remain untouched. The ESPPRC is NP-hard [12] but can be solved heuristically. The proposed column generation heuristic is an adaptation of the exact method developed by Desaulniers et al. [10]. In their method, a sequence of column generators with varying solution times is used at each iteration to generate negative reduced cost variables. At each iteration, the first generator invoked is a tabu search algorithm that rapidly finds negative reduced cost columns in most iterations. When failing to do so, an attempt is made to generate such columns using a heuristic dynamic programming algorithm. If failure occurs again, an exact dynamic programming algorithm, is called to ensure the optimality of the solution process. In the proposed heuristic, both dynamic programming procedures are omitted and the tabu search method is used as the sole column generator. The subproblem and the tabu search method used to solve it are described in Sections 4.1.1 and 4.1.2, respectively.

Two strategies are used to speed up the reconstruction process. First, for large instances (400 customers or more), column generation is prematurely halted when no improvement in the objective value of the RMP has been realized in the last $I_{CG}^{\max}$ column generation iterations ($I_{CG}^{\max}$ was set at 5 for our experiments). Second, the column generation heuristic is warm started at each LNS iteration by introducing, into the first RMP, variables that were generated in previous LNS

iterations and whose corresponding routes are still valid for the current neighborhood. The management of these variables is explained in Section 4.1.3.

4.1.1. Subproblem. As mentioned earlier, the subproblem is an ESPPRC which can be defined over the network $G = (V, \mathcal{A})$ described in the introduction. However, to compute the path with the least reduced cost when solving the subproblem, the costs c_{ij}, $(i,j) \in \mathcal{A}$, are replaced by reduced costs

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \alpha_i & \text{if } i \in \mathcal{N} \\ c_{ij} - \mu & \text{if } i = o, \end{cases}$$

where α_i, $i \in \mathcal{N}$, and μ are the dual variable values of the RMP constraints (9) and (10) at the current column generation iteration, respectively.

Each feasible vehicle route can be represented by a path in G. However, resource constraints (see Ref. [21]) are required in the subproblem to ensure the feasibility of the path with respect to time windows, vehicle capacity, and elementarity. A resource is a quantity that accumulates along a path and is restricted to an interval at each node. For further details about the ESPPRC subproblem, consult Irnich and Desaulniers [21].

4.1.2. Tabu Search. The tabu search method used to solve the ESPPRC was proposed by Desaulniers et al. [10] to quickly generate negative reduced cost columns. Tabu search (see Ref. [17]) is a metaheuristic that has been successful at solving a wide variety of combinatorial optimization problems. It is an iterative method that starts from an initial solution and applies moves to improve it. Possible moves can be defined by a set of operators and are generally quite simple. A neighbor is a solution that differs from a current solution by only one move, and at each iteration, the move creating the best neighbor is chosen even if the objective value is deteriorated. To avoid cycling, a memory of past moves, the tabu list, is kept in order to forbid recent moves to be reversed for a number of iterations. This allows the search to escape from local minima.

For the ESPPRC, the tabu method that we use relies on two operators, inserting a customer in the current route and removing a customer from this route. Each time that a customer is inserted or removed, the reverse move becomes tabu for a fixed number of iterations. Therefore, at each iteration, all possible moves are to remove every (non-tabu) customer individually from the route and to insert every other (non-tabu) customer individually at every possible insertion point in the route. The search space is limited to only feasible routes. Thus, for each insertion move, route feasibility is checked in terms of time windows and vehicle capacity. No feasibility checks are needed for customer removal moves. Since sequences of customers can be fixed with respect to the neighborhood defined by the destruction operators, the tabu method treats these sequences as aggregated customers that cannot be visited separately.

To diversify the search, the tabu search method is started multiple times from a set of different initial solutions and limited to a maximum number $I_{\max}^{\text{tabu,sol}}$ to be performed for each initial solution. The set of initial solutions differs in both phases of our algorithm. In the VNR phase, only the routes associated with non-degenerate variables in the current RMP solution are used. Because the number of these variables can vary, the total number of tabu search iterations per column generation iteration $I_{\max}^{\text{tabu,tot}}$ is fixed and evenly divided among these variables. In the TDR phase, the routes associated with all basic variables (regardless of their value) are used as initial solutions. In both cases, all initial solutions have a reduced cost of 0, and are thus very good initial solutions since we are looking for negative reduced cost values.

4.1.3. Long-Term Column Memory. At the end of each iteration of the LNS algorithm, the columns present in the last RMP solved are added to a pool of columns that can be reused in subsequent iterations. After defining the neighborhood at each iteration, the pool of columns is scanned to find columns that are valid with respect to the neighborhood structure. These columns are added to the RMP to warm start it. This procedure typically reduces the number of column generation iterations and gives access to columns that tabu search may not be able to generate. With these initial columns, column generation is usually faster and can produce higher quality heuristic solutions. However, managing this pool of columns requires a significant amount of computational time at each LNS iteration. This management time often outweighs the time saved by the column generation heuristic when the pool contains a very large number of columns. Therefore, we limit the number of columns in this pool to benefit from them while tempering the resulting pool management time. When the number of columns in memory is greater than a predetermined upper limit at the end of an LNS iteration, columns are eliminated randomly from the pool until a lower limit is reached (set at 70% of the upper limit).

4.2. Branching Strategy

In order to quickly derive integer solutions, we propose to use an effective heuristic branching method. As suggested in Desaulniers et al. [9], we impose decisions on the route variables θ_p and explore the search tree using a depth-first strategy without backtracking. At the end of each linear relaxation, when a fractional solution is found, the route variable with the largest fractional value is simply fixed at 1. No backtracking is allowed, that is, branching decisions cannot be reversed to go up in the search tree. Because of this, the solution found at the end of the branching process might be worst than the previous one or even no feasible solution might be found. In the former case, the deteriorated solution is kept to contribute to the diversification of the search. In the latter case, the previous solution is reused as the current solution.

5. COMPUTATIONAL EXPERIMENTS

We tested our method on the well-known benchmark VRPTW instances of Solomon [30], and Gehring and Homberger [14]. All our tests were performed on an AMD OPTERON processor clocked at 2.3 GHz. The branch-and-price heuristic was implemented using the Gencol software library, version 4.5, (developed at the GERAD research center in Montreal) and relied on the CPLEX solver, version 9.1, for solving the RMP. This section describes the instances, gives the parameter setting, and reports the main results before providing the results of a sensitivity analysis on certain parameter values.

5.1. Instances

Solomon [30] designed 56 VRPTW instances with 100 customers divided into six classes R1, R2, RC1, RC2, C1, and C2, each containing between 8 and 12 instances. In the R class instances, the customers are distributed randomly in the space. In the C class instances, they are clustered. The RC class instances have a mixed distribution of customers. Type 1 instances have a shorter scheduling horizon and thus allow fewer customers to be serviced per route, while Type 2 instances have a longer scheduling horizon. The average width of the time windows and the number of customers with constraining time windows also vary from one instance to another within the same class. Based on the same principles, Gehring and Homberger [14] introduced larger VRPTW instances involving 200, 400, 600, 800, and 1000 customers, with 10 instances in each class for each size.

In the following, instances are identified with ids. Each instance id starts with its class, followed by a number indicating its size, and another number corresponding to the rank of the instance within its class. For example, instance R2_6_9 is the ninth instance of the class R2 with 600 customers.

5.2. Parameter Values

All parameter values have been adjusted through a series of preliminary tests on a subset of the instances. Table 1 provides the parameter values that were not given previously and that were used for the results presented in Subsection 5.3. These parameters are the most sensible ones and will be subject to a sensitivity analysis in Subsection 5.4. Note that for the number of customers removed at each LNS iteration, we used two different values: 70 for the small instances (with 100 and 200 customers) and 100 for the larger ones.

TABLE 1. Parameter values.

Parameter	Instance sizes	Value
Number of customers removed	100–200	70
	400–1,000	100
Maximum number of columns in memory pool	100–1,000	50,000
Number of tabu iterations per column generation iteration in VNR phase ($I_{\max}^{\text{tabu,tot}}$)	100–1,000	$\lvert\mathcal{N}\rvert$
Number of tabu iterations per initial solution in TDR phase ($I_{\max}^{\text{tabu,sol}}$)	100–1,000	5
Maximum number of VNR iterations to find a feasible solution ($I_{\max}^{\text{VNR}}$)	100–1,000	600
Number of TDR iterations ($I_{\max}^{\text{TDR}}$)	100–1,000	800

5.3. Main Results

Tables 2–7 present the results of our algorithm on the benchmark instances and those of the best algorithms that can be found in the literature. For the 100-customer instances (Table 2), these are the algorithms of Pisinger and Ropke [27], Bent and Van Hentenryck [1], Bräysy [3], and Ibaraki et al. [19], which are abbreviated by PR, BVH, B, and IIK-MUY, respectively. For the larger instances (Tables 3–7), the algorithms of Pisinger and Ropke [27], Gehring and Homberger [15], Mester and Bräysy [26], Le Bouthillier et al. [25], abbreviated by PR, GH, MB, and LCK, respectively, serve as a comparison basis. Note that the algorithm of Homberger and Gehring [1] has not been retained because the best results they report were obtained using multiple runs with multiple parameter configurations for each instance [18]. Furthermore, the results of Ibaraki et al. [20] are also not considered because they were derived using the minimum number of vehicles reported in the literature for each instance [20].

For each instance class and each method, we provide in the following tables two values: the mean number of vehicles used and the mean total traveled distance. Rows CNV and CTD show, respectively, the cumulative number of vehicles and the cumulative total distance for all instances in the data set. Row CPU presents the type of processor used and row Time (min), the average time in minutes taken by the algorithm for solving an instance once. The last row, Runs, gives the number of runs performed in order to obtain the results. The results of our algorithm (PDR) are, however, presented in two columns. The first one, Best, indicates the best results obtained in five runs, while the second one, Avg, specifies the average value of the solutions obtained. Bold numbers highlight the best results for each class.

On these main runs, our algorithm succeded to improve the best known solution value of 106 instances (of 356), compared to values reported on the Sintef website[1], as of August 30, 2007, and the website[2] presenting the detailed results of Ibaraki et al. [20]. In all the experiments performed during this project including parameter tuning and sensitivity analysis, a total of 145 new best solutions were found. The values of these new solutions are reported in Table 8. Bold values indicate solutions reducing the number of vehicles compared to the previously best known solutions.

Compared to other methods, our LNS algorithm finds better CNVs and CTDs for all instance sizes, when taking the

[1] www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html
[2] www.al.cm.is.nagoya-u.ac.jp/~yagiura/papers/vrpctw_abst.html

TABLE 2. Solomon's instances with 100 customers.

	PDR					
	Best	Avg	PR (2007)	BVH (2004)	B (2003)	IIKMUY (2005)
R1	**11.92**	11.92	11.92	11.92	11.92	11.92
	1210.34	1211.69	1212.39	1211.10	1222.12	1217.40
R2	2.73	2.85	2.73	**2.73**	2.73	2.73
	955.74	939.861	957.72	**954.27**	975.12	959.11
RC1	**11.50**	11.50	11.50	11.50	11.50	11.50
	1384.16	1386.98	1385.78	1384.17	1389.58	1391.03
RC2	**3.25**	3.28	3.25	3.25	3.25	3.25
	1119.44	1115.68	1123.49	1124.46	1128.38	1122.79
C1	**10.00**	**10.00**	**10.00**	**10.00**	**10.00**	**10.00**
	828.38	**828.38**	**828.38**	**828.38**	**828.38**	**828.38**
C2	**3.00**	**3.00**	**3.00**	**3.00**	**3.00**	**3.00**
	589.86	**589.86**	**589.86**	**589.86**	**589.86**	**589.86**
CNV	**405**	406.6	405	405	405	405
CTD	**57240**	57101	57332	57273	57710	57444
CPU	OPT 2.3 GHz	OPT 2.3 GHz	P4 3 GHz	SU 10	P-200 MHz	P3 1 GHz
Time (min)		30	2.5	120	82.5	250
Runs		5	10	>5	1	1

best of five runs (column Best). For instances with 600 customers or less, we obtained solutions that exhibit the same CNVs as the best known CNVs, but lower CTDs. For the larger instances, the CNVs were significantly reduced (from 2,758 to 2,745 and from 3,438 to 3,432 for the 800- and 1,000-customer instances, respectively). Note, however, that when we obtain the best cumulative results for a given size, we do not obtain the best results for all instance classes. On average (see the Avg columns), the quality of the solutions produced by our algorithm remains very good, especially for the instances with 400 customers or more. Indeed, the average CNVs and CTDs are next to the best for the 400- and 600-customer instances, and the best for the 800- and 1,000-customer instances. These results show that our method is very efficient and robust at finding very good quality solutions for the VRPTW. It is, however, somewhat slow when compared to other methods like that of Pisinger and Ropke [27].

In order to verify the efficiency of the destruction operators, statistics on the performance of the operators were gathered from the main runs. Table 9 provides two values for each operator in each phase for the data sets involving between 400 and 1,000 customers. The first one, nb calls, is the total number of LNS iterations in the corresponding phase in which the given operator was called as the destruction operator, and the second one, nb impr, is the total number of times among these iterations that the current solution was improved. The row Total gives the total numbers of calls

TABLE 3. Gehring and Homberger's instances with 200 customers.

	PDR					
	Best	Avg	PR (2007)	GH (2001)	MB (2005)	LCK (2005)
R1	18.2	18.20	18.2	18.2	18.2	**18.2**
	3615.69	3624.10	3631.226	3885.03	3618.68	**3615.06**
R2	**4.0**	4.02	4.0	4.0	4.0	4.0
	2937.67	2949.28	2949.368	3032.49	2942.92	2969.90
RC1	**18.0**	18.00	18.0	18.1	18.0	18.0
	3192.56	3211.43	3212.282	3674.91	3221.34	3255.97
RC2	4.3	4.38	**4.3**	4.4	4.4	4.3
	2559.32	2540.23	**2556.874**	2671.34	2519.79	2584.18
C1	18.9	18.90	18.9	18.9	**18.8**	18.9
	2718.77	2721.71	2721.522	2842.08	**2717.21**	2736.84
C2	**6.0**	6.00	6.0	6.0	6.0	6.0
	1831.59	1831.88	1832.947	1856.99	1833.57	1833.91
CNV	**694**	695	694	696	694	694
CTD	**168556**	168786	169042	179328	168573	169958
CPU	OPT 2.3 GHz	OPT 2.3 GHz	P4 3 GHz	P-400 MHz	P4 2 GHz	P3 850 MHz
Time (min)		53	7.7	4 × 2.1	8	5 × 10
Runs		5	10	3	1	1

TABLE 4. Gehring and Homberger's instances with 400 customers.

	PDR					
	Best	Avg	PR (2007)	GH (2001)	MB (2005)	LCK (2005)
R1	36.4	36.40	36.4	36.4	**36.3**	36.4
	8420.52	8451.44	8540.04	9478.22	**8530.03**	8607.97
R2	8.0	8.00	8.0	8.0	**8.0**	8.0
	6213.48	6278.74	6241.72	6650.28	**6209.94**	6302.08
RC1	**36.0**	36.00	36.0	36.1	36.0	36.0
	7940.65	8002.87	8069.30	9294.99	8066.44	8267.81
RC2	8.6	8.80	**8.5**	8.8	8.8	8.6
	5269.09	5290.13	**5335.09**	5629.43	5243.06	5397.54
C1	**37.6**	37.62	37.6	38.0	37.9	37.9
	7182.75	7199.78	7290.16	7855.82	7148.27	7223.06
C2	**11.9**	11.98	12.0	12.0	12.0	12.0
	3874.58	3854.17	3844.69	3940.19	3840.85	3862.66
CNV	**1385**	1388.0	1385	1392	1389	1389
CTD	**389011**	390771	393210	428489	390386	396611
CPU	OPT 2.3 GHz	OPT 2.3 GHz	P4 3 GHz	P-400 MHz	P4 2 GHz	P3 850 MHz
Time (min)		89	15.8	4 × 7.1	17	5 × 20
Runs		5	10	3	1	1

and improvements over all datasets, while the last row indicates the percentage of times that each operator improves the current solution when called. From these results, we observe that the behavior of each operator is similar from one instance size to another and that the percentage of improved solutions found is smaller in the TDR phase. Most important, we see that while some operators are more efficient than others, none is of second importance.

To conclude this section, we present another table to compare the effort spent by the algorithm in both phases. For each data set with 400 to 1,000 customers and for each phase, Table 10 specifies the average number of LNS iterations (nb itr) performed (recall that 800 iterations in the TDR phase was set by a parameter), the computational time (in minutes) spent in this phase, and the average time per iteration (time per it). It also reports the average number of instances per data set (each containing 60 instances) for which the VNR phase was stopped because the lower bound m_{lb} on the number of vehicles was reached. From these results, we observe that the average number of iterations executed in the VNR phase is relatively low. This is due to the large number of times (more than 2/3 of the times) that this phase was stopped because the lower bound m_{lb} was reached. In fact, for most instances, the VNR phase is not very time consuming but, for others where it is difficult to reduce the number of vehicles used, it can take up to 50% of the total computational time. For the different

TABLE 5. Gehring and Homberger's instances with 600 customers.

	PDR					
	Best	Avg	PR (2007)	GH (2001)	MB (2005)	LCK (2005)
R1	**54.5**	54.50	54.5	54.5	54.5	54.8
	18252.13	18359.92	18888.52	21864.47	18358.68	18698.37
R2	11.0	11.00	**11.0**	11.0	11.0	11.2
	12808.59	12974.58	**12619.26**	13656.15	12703.52	12989.35
RC1	**55.0**	55.00	55.0	55.0	55.0	55.2
	16266.14	16376.34	16594.94	19114.02	16418.63	16643.27
RC2	11.7	11.94	**11.6**	11.9	12.1	11.8
	10990.85	10926.01	**10777.12**	11670.29	10677.46	10868.94
C1	**57.4**	57.40	57.5	57.7	57.8	57.7
	14106.03	14134.81	14065.89	14817.25	14003.09	14166.80
C2	**17.5**	17.60	17.5	17.8	17.8	17.9
	7632.37	7646.82	7801.296	7889.96	7455.83	7582.61
CNV	**2071**	2074.4	2071	2079	2082	2086
CTD	**800561**	804184	807470	890121	796172	809493
CPU	OPT 2.3 GHz	OPT 2.3 GHz	P4 3 GHz	P-400 MHz	P4 2 GHz	P3 850 MHz
Time (min)		105	18.3	4 × 12.9	40	5 × 30
Runs		5	5	3	1	1

TABLE 6. Gehring and Homberger's instances with 800 customers.

	PDR					
	Best	Avg	PR (2007)	GH (2001)	MB (2005)	LCK (2005)
R1	**72.8**	72.80	72.8	72.8	72.8	72.8
	31797.42	31949.31	32316.79	34653.88	31918.47	32290.48
R2	15.0	15.00	15.0	15.0	**15.0**	15.0
	20651.81	20845.50	20353.51	21672.85	**20295.28**	20765.88
RC1	**72.0**	72.00	73.0	72.3	73.0	72.3
	33170.01	33756.19	29478.3	40532.35	30731.07	37075.19
RC2	15.8	16.12	**15.7**	16.1	15.8	15.8
	16852.38	16927.65	**16761.95**	17941.23	16729.18	17202.08
C1	**75.4**	75.54	75.6	76.1	76.2	76.2
	25093.38	25097.40	25193.13	26936.68	25132.27	25612.47
C2	**23.5**	23.60	23.7	23.7	23.7	24.0
	11569.39	11578.86	11725.46	11847.92	11352.29	11393.80
CNV	**2745**	2750.6	2758	2760	2765	2761
CTD	**1391344**	1401549	1358291	1535849	1361586	1443399
CPU	OPT 2.3 GHz	OPT 2.3 GHz	P4 3 GHz	P-400 MHz	P4 2 GHz	P3 850 MHz
Time (min)		129	22.7	4 × 30.1	145	5 × 40
Runs		5	5	3	1	1

sizes, the time spent in this phase varies between 11% and 22% of the overall time. This can be a bit surprising given the very good results that was obtained for the CNVs. Finally, we note also that, as expected, the average computational time per iteration increases with the size of the instances and that this average time is quite similar in both phases.

5.4. Sensitivity Analysis

Tests were made to verify the behaviour of our algorithm with regards to variations of certain parameter values. For each instance size from 400 to 1000 customers, a subset sub-(size) of instances was selected to cover a variety of instance types. The subset sub-400 consists of the following instances: R1_4_9, R2_4_1, RC1_4_4, RC2_4_2, RC2_4_5, RC2_4_6, C1_4_7, C1_4_8, C2_4_2, and C2_4_3. For the other sizes, the susbets contain the corresponding instances (R1_6_9, R2_6_1, …).

The sensitivity analysis was performed for each parameter in Table 1, where one parameter value was changed at once. The test results can be found in Tables 11–16. As in Section 5.3, five runs were performed for each instance and, for each subset, two values are reported, namely, the mean number of vehicles used and the mean total distance. CNV and CTD values are given as well and correspond to the sums over all the subsets. For comparison purposes, the last column provides the values of the best published solutions. Finally, the last row of each table gives the average computation time (in minutes) for solving an instance.

TABLE 7. Gehring and Homberger's instances with 1000 customers.

	PDR					
	Best	Avg	PR (2007)	GH (2001)	MB (2005)	LCK (2005)
R1	91.9	91.90	92.2	91.9	92.1	92.0
	49702.32	50168.00	50751.25	58069.61	49281.48	51847.22
R2	19.0	19.00	**19.0**	19.0	19.0	19.0
	30495.26	30730.35	**29780.82**	31873.62	29860.32	30441.05
RC1	90.0	90.00	90.0	90.1	**90.0**	90.0
	45574.11	45924.74	46752.15	50950.14	**45396.41**	46118.08
RC2	18.5	18.82	**18.3**	18.5	18.7	18.5
	25470.33	25464.40	**25090.88**	27175.98	25063.51	25390.40
C1	**94.3**	94.50	94.6	95.4	95.1	95.1
	41783.27	41913.42	41877.00	43392.59	41569.67	42403.21
C2	**29.5**	29.56	29.7	29.7	29.7	29.6
	16657.06	16817.76	16840.37	17572.72	16639.54	17164.51
CNV	**3432**	3437.8	3438	3446	3446	3442
CTD	**2096823**	2110187	2110925	2290367	2078110	2133645
CPU	OPT 2.3 GHz	OPT 2.3 GHz	P4 3 GHz	P-400 MHz	P4 2 GHz	P3 850 MHz
Time (min)		162	26.6	4 × 30.1	600	5 × 50
Runs		5	5	3	1	1

TABLE 8. New best solution values.

	R1		R2		RC1		RC2		C1		C2	
#	Veh	Dist	Veh	Dist	Veh	Dist	Veh	Dist	Veh	Dist	Veh	Dist
200 customers												
1	—	—	4	4483.16	18	3602.80	6	3099.53	—	—	—	—
2	18	4040.60	4	3621.20	18	3249.50	5	2825.24	—	—	—	—
3	—	—	—	—	18	3008.76	4	2603.83	18	2707.35	—	—
4	18	3059.22	—	—	—	—	—	—	18	2643.97	6	1703.43
5	18	4107.86	4	3366.79	18	3385.88	4	2911.46	—	—	—	—
6	18	3583.14	4	2913.03	18	3324.80	—	—	—	—	—	—
7	18	3150.11	—	—	18	3189.32	4	2526.18	—	—	—	—
8	18	2952.65	—	—	18	3083.93	4	2293.35	—	—	—	—
9	18	3760.58	4	3092.53	—	—	—	—	—	—	—	—
10	18	3302.72	—	—	18	3008.53	—	—	—	—	—	—
400 customers												
1	—	—	8	9213.68	36	8630.94	11	6688.31	—	—	—	—
2	36	8955.50	8	7641.67	36	7958.67	—	—	36	7687.38	—	—
3	36	7841.52	—	—	36	7562.60	8	4958.74	36	7060.73	**11**	**4109.88**
4	36	7318.62	8	4297.20	36	7332.59	—	—	—	—	**11**	**3865.45**
5	36	9242.43	—	—	36	8249.63	9	5923.95	—	—	—	—
6	36	8383.67	—	—	36	8223.12	8	5863.56	—	—	—	—
7	36	7656.94	—	—	36	8001.12	8	5466.70	39	7417.92	—	—
8	36	7293.69	—	—	36	7836.29	8	4848.87	37	7363.51	—	—
9	36	8750.84	—	—	36	7811.55	—	—	36	7061.21	12	3865.65
10	36	8125.03	—	—	36	7668.77	8	4311.59	36	6860.63	**11**	**4115.46**
600 customers												
1	—	—	11	18291.18	55	17317.13	—	—	—	—	—	—
2	54	19147.38	—	—	55	16123.40	—	—	56	14163.31	17	8380.49
3	54	17216.16	—	—	55	15358.13	—	—	56	13781.19	17	7595.43
4	54	15947.03	—	—	—	—	—	—	—	—	—	—
5	54	20017.80	—	—	55	16934.45	12	12168.79	—	—	—	—
6	54	18237.76	—	—	55	16842.27	—	—	—	—	18	7472.24
7	54	16796.63	—	—	55	16450.42	—	—	58	14851.65	18	7512.33
8	—	—	—	—	55	16164.82	—	—	**56**	**14541.53**	17	7778.30
9	54	19015.51	—	—	—	—	—	—	56	13718.23	—	—
10	54	18204.18	—	—	55	15936.81	—	—	56	13669.88	—	—
800 customers												
1	—	—	15	28392.87	**72**	**35102.79**	19	20520.49	—	—	—	—
2	72	33190.68	—	—	72	33361.67	—	—	74	25528.55	23	12332.37
3	72	29943.87	—	—	72	30608.16	—	—	72	24366.83	23	11438.72
4	—	—	—	—	72	28363.65	—	—	—	—	—	—
5	72	34247.99	—	—	**72**	**34481.02**	16	18917.65	—	—	—	—
6	72	31728.99	—	—	**72**	**34849.96**	15	18600.22	—	—	—	—
7	72	29399.21	—	—	72	33102.75	—	—	**77**	**26639.13**	24	11380.54
8	72	28191.89	—	—	72	33188.75	—	—	**74**	**25370.02**	—	—
9	72	33074.30	—	—	72	33350.51	—	—	72	24698.05	23	12301.63
10	—	—	—	—	72	31766.56	—	—	72	24324.76	23	11163.89
1,000 customers												
1	100	53904.23	—	—	—	—	—	—	—	—	—	—
2	91	50701.78	—	—	—	—	—	—	—	—	—	—
3	91	46169.17	—	—	90	43390.58	—	—	—	—	—	—
4	—	—	—	—	—	—	—	—	—	—	—	—
5	91	54032.44	—	—	90	46631.89	—	—	—	—	—	—
6	—	—	—	—	—	—	—	—	—	—	—	—
7	91	45729.79	—	—	—	—	—	—	**98**	**42824.09**	—	—
8	—	—	—	—	90	45406.46	—	—	**93**	**42499.59**	—	—
9	—	—	—	—	90	45149.72	—	—	**90**	**41318.12**	**29**	**16751.82**
10	—	—	—	—	90	44947.71	—	—	90	40586.60	—	—

TABLE 9. Use of different operators.

	Route portion				Proximity			
	VNR		TDR		VNR		TDR	
Instance size	nb calls	nb impr	nb calls	nb impr	nb calls	nb impr	nb calls	nb impr
400	9,479	2,367	56,962	13,532	14,191	4,231	73,094	17,747
600	8,298	2,114	56,347	13,885	13,814	4,600	77,922	19,300
800	13,651	3,522	52,284	13,651	21,481	8,241	81,312	22,361
1000	13,552	4,563	56,197	16,683	23,851	9,277	79,056	23,851
Total	43,056	12,566	221,790	57,751	71,166	26,349	311,384	83,259
%		29.2		26.0		37.0		26.7
	Longest detour				Time			
	VNR		TDR		VNR		TDR	
Instance size	nb calls	nb impr	nb calls	nb impr	nb calls	nb impr	nb calls	nb impr
400	12,388	3,761	52,751	10,697	11,100	3,501	57,193	12,438
600	9,694	3,341	47,047	8,251	11,233	4,307	58,684	12,388
800	15,012	5,616	45,587	7,764	15,609	6,492	60,817	14,031
1000	14,949	6,197	42,044	8,084	15,937	7,156	62,703	16,463
Total	52,043	18,915	187,429	34,796	53,879	21,456	239,397	55,320
%		36.3		18.6		39.8		23.1

The results reported in Tables 11–16 are discussed in the following paragraphs, where the emphasis is put on the CNV and CTD values, as well on the computational times. Indeed, in most cases, the results for the different subsets are very consistent with the cumulative values.

5.4.1. Number of Customers Removed (Table 11). The results indicate that using larger neighborhoods yields better solutions, but takes longer computational times. Removing 100 customers is however sufficient to obtain the same CNV as that of the best published solutions and requires much less computational time than the case where 120 cutomers are removed.

TABLE 10. Statistics on the two phases.

	VNR				TDR		
Instance size	nb it	tot time	Time per it	m_{lb} reached	nb itr	tot time	Time per it
400	158	9	0.06	40.2	800	81	0.10
600	144	13	0.09	44.8	800	92	0.11
800	214	26	0.12	38.8	800	103	0.13
1000	221	35	0.16	39.6	800	127	0.16

5.4.2. Maximum number of columns in memory pool (Table 12). To a certain extent, keeping more columns in the memory pool helps computing better solutions. It however requires longer computational times to manage a larger number of columns. Notice that keeping no columns at all in memory yields very bad quality solutions.

5.4.3. Number of Tabu Search Iterations per Column Generation Iteration in the VNR Phase (Table 13). On average, allowing more tabu search iterations per column generation iteration in the VNR phase helps reducing the number of vehicles used. However, with $1.3|\mathcal{N}|$ tabu search iterations, it was not possible to find 1421 vehicles among the best solutions as it was the case with $1.0|\mathcal{N}|$ iterations. This is due to the heuristic nature of the method. It should be noted that increasing the value of this parameter slowly increases computational times, because the number of LNS iterations performed in the VNR phase accounts for ∼11–17% of the total number of LNS iterations (see Table 10).

5.4.4. Number of Tabu Search Iterations per Initial Solution in the TDR Phase (Table 14). As expected, increasing the value of this parameter reduces the total distance. On the other hand, it increases computational times rather rapidly.

5.4.5. Maximum Number of VNR Iterations to Find a Feasible Solution (Table 15). Increasing the value of this parameter can only improve the solution quality in terms of

TABLE 11. Sensitivity analysis on the number of customers removed.

	80		100		120		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.8	20.90	20.7	20.88	20.7	20.86	20.6
	6551.27	6582.42	6587.39	6572.04	6574.22	6549.16	6651.14
sub-600	30.7	30.94	30.4	30.64	30.4	30.66	30.4
	13414.03	13332.73	13557.75	13462.11	13475.26	13413.95	13412.17
sub-800	40.6	40.94	40.5	40.86	40.5	40.74	40.6
	22536.34	22492.23	22379.96	22363.30	22305.19	22342.87	22398.83
sub-1000	50.6	51.04	50.5	50.80	50.5	50.72	50.5
	34446.49	34391.39	34261.47	34378.06	34147.35	34202.54	33694.66
CNV	1,427	1438.2	1,421	1,431.8	1,421	1429.8	1,421
CTD	769,481	767,987	767,866	767,755	765,020	765,085	761568
Time (min)		88		143		218	

TABLE 12. Sensitivity analysis on the maximum number of columns in memory pool.

	0		20,000		50,000		100,000		
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	21.0	21.10	20.8	20.92	20.7	20.88	20.7	20.72	20.6
	6636.47	6691.53	6583.66	6592.47	6587.39	6572.04	6566.48	6598.17	6651.14
sub-600	31.0	31.06	30.8	30.94	30.4	30.64	30.5	30.72	30.4
	13334.99	13514.01	13318.26	13379.34	13557.75	13462.11	13407.66	13351.09	13412.17
sub-800	41.0	41.32	40.6	40.98	40.5	40.86	40.6	40.90	40.6
	22693.96	22691.46	22665.14	22568.92	22379.96	22363.30	22295.38	22299.63	22398.83
sub-1000	51.0	51.36	50.8	50.98	50.5	50.80	50.6	50.82	50.5
	34828.89	34902.62	34332.58	34615.52	34261.47	34378.06	34032.07	34237.34	33694.66
CNV	1,440	1448.4	1,430	1438.2	1,421	1431.8	1,424	1431.6	1,421
CTD	774,943	777,996	768,996	771,563	767,866	767,755	763,016	764,862	761,568
Time (min)		75		117		143		170	

TABLE 13. Sensitivity analysis on the number of tabu search iterations per column generation iteration in the VNR phase.

	$0.7\|\mathcal{N}\|$		$1.0\|\mathcal{N}\|$		$1.3\|\mathcal{N}\|$		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.9	20.94	20.7	20.88	20.8	20.90	20.6
	6510.76	6539.91	6587.39	6572.05	6544.16	6550.11	6651.14
sub-600	30.6	30.84	30.4	30.64	30.4	30.70	30.4
	13329.61	13325.91	13557.75	13462.11	13494.42	13404.53	13412.17
sub-800	40.6	40.96	40.5	40.86	40.5	40.74	40.6
	22324.69	22289.15	22379.96	22363.30	22295.51	22386.59	22398.83
sub-1000	50.6	50.90	50.5	50.80	50.6	50.80	50.5
	34313.08	34347.92	34261.47	34378.06	34234.19	34341.86	33694.66
CNV	1,427	1436.4	1,421	1431.8	1,423	1431.4	1,421
CTD	764,781	765,029	767,866	767,755	765,683	766,831	761,568
Time (min)		134		143		150	

the number of vehicle used. With 800 iterations, we even succeeded to obtain 1,420 vehicles which is better than the best results reported in the literature. Note also that increasing the value of this parameter slowly increases computational times, because the VNR phase is often stopped when the lower bound m_{lb} on the number of vehicles is reached.

5.4.6. Number of Iterations in the TDR Phase (Table 16). Increasing the value of this parameter cannot deteriorate the solution quality. In fact, it helps reducing the total distance as shown by the results. Computational times however increase rather rapidly with the number of iterations in the TDR phase.

TABLE 14. Sensitivity analysis on the number of tabu search iterations per initial solution in the TDR phase.

	3		5		10		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.7	20.88	20.7	20.88	20.7	20.88	20.6
	6603.70	6594.60	6587.39	6572.05	6598.67	6582.22	6651.14
sub-600	30.4	30.64	30.4	30.64	30.4	30.64	30.4
	13579.14	13494.57	13557.75	13462.11	13548.48	13462.88	13412.17
sub-800	40.5	40.86	40.5	40.86	40.5	40.86	40.6
	22531.60	22513.82	22379.96	22363.30	22306.04	22333.11	22398.83
sub-1000	50.5	50.80	50.5	50.80	50.5	50.80	50.5
	34303.16	34450.05	34261.47	34378.06	34109.92	34282.22	33694.66
CNV	1,421	1431.8	1,421	1,431.8	1,421	1431.8	1,421
CTD	770,176	770,530	767,866	767,755	765,631	766,604	761,568
Time (min)		120		143		184	

TABLE 15. Sensitivity analysis on the maximum number of VNR iterations to find a feasible solution.

	400		600		800		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.7	20.90	20.7	20.88	20.6	20.82	20.6
	6596.57	6566.46	6587.39	6572.04	6632.94	6585.08	6651.14
sub-600	30.6	30.72	30.4	30.64	30.4	30.62	30.4
	13373.38	13410.48	13557.75	13462.11	13543.40	13458.55	13412.17
sub-800	40.5	40.90	40.5	40.86	40.5	40.80	40.6
	22350.63	22318.69	22379.96	22363.30	22384.03	22439.04	22398.83
sub-1000	50.6	50.90	50.5	50.80	50.5	50.78	50.5
	34218.14	34298.80	34261.47	34378.06	34295.44	34427.53	33694.66
CNV	1,424	1434.4	1,421	1431.8	1,420	1430.2	1,421
CTD	765,387	765,944	767,866	767,755	768,558	769,102	761,568
Time (min)		131		143		152	

TABLE 16. Sensitivity analysis on the number of iterations in the TDR phase.

	400		800		1200		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.7	20.88	20.7	20.88	20.7	20.88	20.6
	6629.36	6604.31	6587.39	6572.05	6580.95	6559.11	6651.14
sub-600	30.4	30.64	30.4	30.64	30.4	30.64	30.4
	13683.35	13577.36	13557.75	13462.11	13486.45	13402.23	13412.17
sub-800	40.5	40.86	40.5	40.86	40.5	40.86	40.6
	22618.20	22687.86	22379.96	22363.30	22289.87	22225.88	22398.83
sub-1000	50.5	50.80	50.5	50.80	50.5	50.80	50.5
	34575.54	34804.44	34261.47	34378.06	34114.23	34189.77	33694.66
CNV	1,421	1431.8	1,421	1431.8	1,421	1431.8	1,421
CTD	775,064	776,740	767,866	767,755	764,715	763,770	761,568
Time (min)		100		143		185	

To conclude this section, we would like to highlight the fact that our algorithm involves close to ten parameters, which might seem a lot to adjust carefully. However, most of them are used to limit computational times in one way or another. The sensitivity analysis results presented in this section clearly show that our algorithm could have found better solutions if we have allowed more time to solve the instances. Hence, we had to make a compromise between solution quality and computational time. We believe that the parameter values used to produce the main results reported in Subsection 5.3 offer a good trade-off between these two criteria.

6. CONCLUSION

In this article, we proposed a new LNS method for solving the VRPTW. This method takes advantage of the power of branch-and-price, the leading methodology for the exact solution of the VRPTW, to efficiently explore the neighborhoods. With this methodology, we succeded to find 145 new best solutions on the well-known 356 benchmark instances of Solomon [30] and Gehring and Homberger [14] and to obtain for all instance sizes the best cumulative number of vehicles (CNVs) and cumulative total distances (CTDs) compared to the results of the best know methods. Furthermore, we demonstrated through a sensitivity analysis on the parameters limiting computational time that our method could produce better solutions if additional computational time was used. Hence, the parameter setting used for our experiments offered a compromise between solution quality and computational time. This compromise led to acceptable computational times which are not as fast as those of certain other leading methods.

Combining branch-and-price and LNS is a relatively new idea that can be applied for a wide variety of vehicle and crew scheduling problems. This paper has shown that such a hybrid methodology can perform very well against the leading heursitic methods for a well-studied problem.

REFERENCES

[1] R. Bent and P. Van Hentenryck, A two-stage hybrid local search for the vehicle routing problem with time windows, Trans Sci 38 (2004), 515–530.

[2] J. Berger, M. Barkaoui, and O. Bräysy, A route-directed hybrid genetic approach for the vehicle routing problem with time windows, INFOR 41 (2003), 179–194.

[3] O. Bräysy, A reactive variable neighborhood search for the vehicle routing problem with time windows, INFORMS J Comput 15 (2003), 347–368.

[4] O. Bräysy and M. Gendreau, Vehicle routing problem with time windows, Part I: Route construction and local search algorithms, Transp Sci 39 (2005), 104–118.

[5] O. Bräysy and M. Gendreau, Vehicle routing problem with time windows, Part II: Metaheuristics, Transp Sci 39 (2005), 119–139.

[6] A. Chabrier, Vehicle routing problem with elementary shortest path based column generation, Comput Oper Res 33 (2006), 2972–2990.

[7] J.-F. Cordeau, G. Laporte, and A. Mercier, A unified tabu search heuristic for the vehicle routing problems with time windows, J Oper Res Soc 52 (2001), 928–936.

[8] R. De Franceschi, M. Fischetti, and P. Toth, A new ILP-based refinement heuristic for vehicle routing problems, Math Program B 105 (2006), 471–499.

[9] G. Desaulniers, J. Desrosiers, and M.M. Solomon, "Accelerating strategies for column generation methods in vehicle routing and crew scheduling problems," Essays and Surveys in Metaheuristics, C. C. Ribeiro and P. Hansen (Editors), Kluwer, Norwell, MA, 2002, pp. 309–324.

[10] G. Desaulniers, F. Lessard, and A. Hadjar, Tabu search, generalized k-path inequalities, and partial elementarity for the vehicle routing problem with time windows, Les Cahiers du GERAD, G-2006-45, HEC Montréal, Montréal, Canada, 2006.

[11] M. Desrochers, J. Desrosiers, and M.M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows, Oper Res 40 (1992), 342–534.

[12] M. Dror, Note on the complexity of the shortest path models for column generation in VRPTW, Oper Res 42 (1994), 977–978.

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

[14] H. Gehring and J. Homberger, "A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows", Proceeding of EUROGEN99 - Short Course on Evolutionary Algorithms in Engineering and Computer Science, K. Miettinen, M. Makela, and J. Toivanen (Editors), University of Jyväskylä, Jyväskylä, Finland, 1999, pp. 57–64.

[15] H. Gehring and J. Homberger, A parallel two-phase metaheuristic for routing problems with time windows, Asia Pac J Oper Res 18 (2001), 35–47.

[16] M. Gendreau, A. Hertz, G. Laporte, and M. Stan, A generalized insertion heuristic for the traveling salesman problem with time windows, Oper Res 46 (1995), 330–335.

[17] F. Glover and M. Laguna, Tabu search, Kluwer, Norwell, MA, 1997.

[18] J. Homberger and H. Gehring, A two-phase hybrid metaheuristic for the vehicle routing problem with time windows, Eur J Oper Res 162 (2005), 220–238.

[19] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura, Effective local search algorithms for routing and scheduling problems with general time window constraints, Transp Sci 39 (2005), 206–232.

[20] T. Ibaraki, S. Imahori, K. Nonobe, K. Sobue, T. Uno, and M. Yagiura, An iterated local search algorithm for the vehicle routing problem with convex time penalty functions, Discrete Appl Math (2007).

[21] S. Irnich and G. Desaulniers, "Shortest path problems with resource constraints," Column Generation, G. Desaulniers, J. Desrosiers, and M. M. Solomon (Editors), Springer, New-York, NY, 2005, pp. 33–65.

[22] S. Irnich and D. Villeneuve, The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$, INFORMS J Comput 18 (2006), 391–406.

[23] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger, 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.

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

[25] A. Le Bouthillier, T.G. Crainic, and P. Kropf, A guided cooperative search for the vehicle routing problem with time windows, IEEE Intell Syst 20 (2005), 36–42.

[26] D. Mester and O. Bräysy, Active guided evolution strategies for large scale vehicle routing problem with time windows, Comput Oper Res 32 (2005), 1592–1614.

[27] D. Pisinger and S. Ropke, A general heuristic for vehicle routing problems, Comput Oper Res 34 (2007), 2403–2435.

[28] L.-M. Rousseau, M. Gendreau, and G. Pesant, Using constraint-based operators to solve the vehicle routing problem with time windows, J Heuristics 8 (2002), 43–58.

[29] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems", Proceedings of Constraints Programming '98, M. Maher and J.F. Puget (Editors), Springer-Verlag, Berlin, 1998, pp. 417–431.

[30] M.M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, Oper Res 35 (1987), 254–265.