Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier

Working Paper DPO-2017-08 (version 1, 23.12.2017)

Dominik Goeke
Deutsche Post Chair – Optimization of Distribution Networks
RWTH Aachen University
goeke@dpo.rwth-aachen.de

Timo Gschwind
Chair of Logistics Management
Johannes Gutenberg University Mainz
gschwind@uni-mainz.de

Michael Schneider
Deutsche Post Chair – Optimization of Distribution Networks
RWTH Aachen University
schneider@dpo.rwth-aachen.de

Abstract

The vehicle-routing problem with private fleet and common carrier (VRPPC) extends the capacitated VRP by considering the option of outsourcing customers to subcontractors at a customer-dependent cost instead of serving them with the private fleet. The VRPPC has important applications in small package shipping and manufacturing, but despite its relevance, no exact solution approach has been introduced so far. We propose a branch-and-price-cut algorithm that is able to solve small to medium-sized instances and provides tight lower bounds for larger instances from the literature. In addition, we develop a large neighborhood search that is competitive with the state-of-the-art on instances assuming a homogeneous vehicle fleet. On the instances assuming a heterogeneous fleet, we are able to improve the previous best known solutions for the large majority of available instances.

Keywords: *vehicle routing, subcontracting, metaheuristic, large neighborhood search, column generation, branch-and-price-and-cut*

1 Introduction

The vehicle-routing problem with private fleet and common carrier (VRPPC) is a variant of the VRP in which customers can be subcontracted at a customer-dependent cost if the privately-owned capacity is insufficient to serve all customers, or if doing so is beneficial from a cost point of view. Consequently, the subcontracted customers do not need to be served on vehicle routes of the privately-owned fleet, but a cost is paid for outsourcing customers to the so-called common carrier.

The VRPPC has direct applications in manufacturing (Tang and Wang 2006) and less-than-truckload shipping (Chu 2005, Stenger et al. 2013a). It is also closely related to problems arising in collaborative transportation, in which carriers can pass on requests to other carriers and accept or decline requests offered by their partners (Liu et al. 2010), and to the integrated operational transportation planning problem that considers different subcontracting options (Krajewska and Kopfer 2009). The VRPPC also has applications in the planning of same-day parcel deliveries. By choosing adequate customer-dependent outsourcing costs, important customers—e.g., subscribers of Amazon Prime or customers that have already been postponed on previous days—can be favored over regular new requests.

Despite its practical relevance, relatively few papers have focused on solution methods for the VRPPC. The following heuristic paradigms have been proposed in the literature: simple construction and improvement heuristics (Chu 2005, Bolduc et al. 2007), randomized construction–improvement–perturbation (RIP, Bolduc et al. 2008), tabu search (TS, Côté and Potvin 2009, Potvin and Naud 2011), variable neighborhood search (VNS, Stenger et al. 2013a,b), multi-start local search (MS-LS, Vidal et al. 2016), iterated local search (MS-ILS, Vidal et al. 2016), and a memetic algorithm (MA, Vidal et al. 2016). All of the listed methods have been investigated on instances assuming a homogeneous vehicle fleet, and the MA of Vidal et al. (2016) shows the best performance with regards to solution quality. On instances with a heterogeneous fleet composition, only RIP and the two TS algorithms have been tested.

It is notable that early approaches (Chu 2005, Bolduc et al. 2007) only take the decision which customers to subcontract into account when constructing the initial solution. Later approaches consider the subcontracting decision when generating the neighborhood of a solution (Bolduc et al. 2008, Côté and Potvin 2009, Potvin and Naud 2011, Stenger et al. 2013b,a). The most recent approach (Vidal et al. 2016) uses an implicit customer selection, i.e., for every move a resource-constrained shortest path problem is solved to evaluate which customers should be subcontracted.

The contribution of this paper is twofold and concerns the heuristic and exact domain:

- We develop a large neighborhood search (LNS) to solve the VRPPC heuristically. Our LNS features a new decomposition procedure, and we demonstrate the effectiveness of this component by comparing it to an LNS without this component on the VRPPC benchmark instances from the literature. We are able to provide several new best-known solutions on the larger instances from the literature, and we demonstrate that our heuristic is among the best solution methods published for the VRPPC.
- To the best of our knowledge, we are the first to propose an exact solution method for the VRPPC. Our method uses a path-based formulation that is solved by means of a branch-price-and-cut algorithm (BPC). The BPC is able to provide optimal solutions for some small to medium-sized instances. For larger instances, it provides tight lower bounds that can be used to assess the quality of the heuristic solutions.

Our paper is organized as follows: Section 2 formally describes the problem. Our two solution methods, BPC and LNS, are explained in Sections 3 and 4. Section 5 details the parameter setting, the test instances and presents the numerical results. Section 6 concludes the paper.

2 The vehicle-routing problem with private fleet and common carrier

To define the VRPPC as a graph-theoretical problem, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a complete undirected graph with vertices $\mathcal{V} = \{v_0\} \cup N$ and edges $\mathcal{E} = \mathcal{V} \times \mathcal{V}$. Vertex v_0 denotes the depot, the other vertices represent customers $i \in \mathcal{N}$. Each customer $i \in \mathcal{N}$ is assigned a demand q_i and a cost h_i for subcontracting the customer. Each edge $\{i, j\} \in \mathcal{E}$ is assigned a travel cost c_{ij}. At the depot, a set of vehicles $\mathcal{K}$, which represent the private fleet, is based. The vehicles $k \in \mathcal{K}$ can differ with regard to capacity C_k and fixed cost F_k. The fixed cost F_k is only incurred if a route is assigned to vehicle k. Typically, not all vehicles are different, and we can group the vehicles according to their attributes such that vehicles with identical attributes are in the same group $l \in \mathcal{L}$. We denote the number of vehicles in group l as z_l. The VRPPC now calls for: i) satisfying the demand of every customer with exactly one visit, either using the common carrier or a vehicle of the private fleet, and ii) planning at most one route for each vehicle of the private fleet so that every route starts and ends at the depot, and the vehicle capacity is respected. The goal is to minimize the total cost consisting of the sum of fixed cost, the cost of routing the vehicles of the private fleet, and the cost of subcontracting customers.

3 Branch-price-and-cut algorithm for the VRPPC

In this section, we give details on our exact approach to the VRPPC that is used to obtain optimal solutions for small to medium-sized instances. For larger instances, we provide lower bounds, which can be used to assess the quality of the heuristic solutions. The approach is based on a path-based formulation that is solved by means of a BPC algorithm.

3.1 Path-based formulation

Master program Let Ω_l be the set of all feasible routes for vehicle group $l \in \mathcal{L}$. We denote by c_r the routing cost of route $r \in \Omega_l$. Binary decision variables λ_r indicate if the route $r \in \Omega_l$ is selected ($\lambda_r = 1$) or not ($\lambda_r = 0$) while binary decision variables y_i indicate if customer $i \in \mathcal{N}$ is served by the common carrier ($y_i = 1$) or by the private fleet ($y_i = 0$). Finally, let a_{ri} be the number of times route r visits customer i. The VRPPC can then be defined as:

$$\min \quad \sum_{l \in \mathcal{L}} \sum_{r \in \Omega_l} (F_l + c_r)\lambda_r + \sum_{i \in \mathcal{N}} h_i y_i \tag{1a}$$

$$\text{s.t.} \quad \sum_{l \in \mathcal{L}} \sum_{r \in \Omega_l} a_{ri}\lambda_r + y_i = 1 \qquad \forall i \in \mathcal{N} \tag{1b}$$

$$\sum_{r \in \Omega_l} \lambda_r \leq z_l \qquad \forall l \in \mathcal{L} \tag{1c}$$

$$\lambda_r \in \{0, 1\} \qquad \forall l \in \mathcal{L}, r \in \Omega_l \tag{1d}$$

$$y_i \in \{0, 1\} \qquad \forall i \in \mathcal{N} \tag{1e}$$

The objective function (1a) minimizes the total cost comprising vehicle fixed costs, routing costs, and subcontracting costs. Partitioning constraints (1b) ensure that each customer is served exactly once either by the private fleet or by the common carrier. Note that constraints (1b) can be replaced by their covering counterpart if routing costs satisfy the triangle inequality. Convexity constraints (1c) limit the number of vehicles of each type. The variable domains are specified in (1d) and (1d).

Because of the huge number of feasible routes, model (1) cannot be solved directly and one has to resort to column-generation (or Lagrangean-relaxation) based methods: The linear relaxation of model (1) is initialized with a proper subset of routes and missing routes (=columns) with negative reduced costs are dynamically identified by calling the pricing subproblems and added to the master program (1). Integrality is finally ensured by integrating the column-generation process into a branch-and-bound algorithm (Lübbecke and Desrosiers 2005).

Pricing subproblem The task of the pricing subproblems is to find feasible routes with negative reduced costs or to prove that no such routes exist. Similar to many other VRP variants, the pricing subproblems of the VRPPC are elementary shortest-path problems with resource constraints (ESPPRCs) on graphs with negative-cost cycles (Irnich and Desaulniers 2005). In the VRPPC, there are $|\mathcal{L}|$ different pricing problems, one for each vehicle type $l \in \mathcal{L}$. Given a vehicle type $l \in \mathcal{L}$, a feasible VRPPC route starts and ends at the depot v_0, visits some customers $i \in \mathcal{N}$ in between, and respects the vehicle capacity C_l.

Let π_i and μ_l be the dual prices of constraints (1b) and (1c), respectively. The reduced cost of a route $r \in \Omega_l$ is given by

$$\tilde{c}_r = c_r - \sum_{i \in \mathcal{N}} \pi_i - \mu_l = \sum_{(i,j) \in \mathcal{E}(r)} \tilde{c}^l_{ij}, \tag{2}$$

where $\mathcal{E}(r)$ denotes the sequence of edges traversed by route r and $\tilde{c}^l_{ij} = c_{ij} - 1/2\tilde{\pi}_i - 1/2\tilde{\pi}_j$ with $\tilde{\pi}_{v_0} = \mu_l$ and $\tilde{\pi}_i = \pi_i$ for all customers $i \in \mathcal{N}$. The pricing subproblem for vehicle type $l \in \mathcal{L}$ can then be formalized as

$$\min_{r \in \Omega_l} \{\tilde{c}_r\}. \tag{3}$$

It is well-known that the ESPPRCs (3) are strongly $\mathcal{NP}$-hard. To obtain better-solvable pricing subproblems, the elementarity condition of routes can be relaxed so that routes containing cycles can be priced out. This comes at the cost of weaker lower bounds of formulation (1). A good trade-off between the hardness of the pricing subproblems and the strength of the lower bounds is often achieved by the so-called ng-routes (Baldacci et al. 2011) that forbid certain types of cycles. Here, each customer $i \in \mathcal{N}$ is assigned a neighborhood $N_i \subset \mathcal{N}$ with $i \in N_i$. Typically, the cardinalities $|N_i|$ and the neighborhoods N_i themselves are fixed a priori for all customers $i \in \mathcal{N}$. An ng-route now allows multiple visits to a customer i provided that another customer j with $i \notin N_j$ is visited in between, i.e., an ng-route forgets previous visits to those customers that are not in the neighborhoods of the subsequently visited customers. In the following, we redefine the set Ω_l as the set of all ng-feasible routes. Clearly, the cycles that are allowed and, thus, the quality of the lower bounds provided by the ng-route relaxation of (1) critically depends on the choices of N_i. Elementarity of all routes of a solution is finally ensured by branching.

3.2 Cutting planes

Path-based models like the extended set-partitioning formulation (1) generally provide much stronger bounds compared to edge-based formulations. Still, even with large $|N_i|$ or pure elementary sets Ω_l, the lower bounds provided by (1) are not sufficiently tight for the effective solution of even small to medium-sized instances of the VRPPC. To further strengthen the formulation, we add the following additional families of valid inequalities.

Robust cuts The first type of cuts describe inequalities on the aggregated flow on edges $\{i,j\} \in \mathcal{E}$. Such inequalities can be incorporated into the master problem using expressions $x(\delta(S)) \leq rhs$ or $x(\delta(S)) \geq rhs$, where $\delta(S) = \{\{i,j\} \in \mathcal{E} : i \in S, j \in \mathcal{N} \setminus S\}$ denotes the cut-set of $S \subset \mathcal{N}$ and $x(\delta(S)) = \sum_{l\in\mathcal{L}} \sum_{r\in\Omega_l} \sum_{\{i,j\}\in\delta(S)} b_{ijr}\lambda_r$ denotes the cut-set flow. Parameter b_{ijr} gives the number of times edge $\{i,j\}$ is traversed by route r. The dual prices of these inequalities directly transfer to the reduced cost $\tilde{c}^l_{ij}, l \in \mathcal{L}$ of the included edges meaning that they do not change the structure and the complexity of the pricing subproblems, i.e., they are *robust cuts*. In our BPC approach, we use rounded capacity cuts, which are separated with the CVRPSEP package (Lysgaard 2003).

Non-robust cuts To further strengthen the linear relaxation, we also incorporate *non-robust* cuts into model (1). The addition of non-robust cuts has to be done carefully because each cut makes the pricing subproblem harder. Subset-row inequalities (SR) originally introduced by Jepsen et al. (2008) are a family of non-robust cuts that have been successfully used in many exact approaches to VRP variants. Each SR is defined on a subset of customers. As proposed by Jepsen et al. (2008), we restrict ourselves to SR defined on three customers because they can be separated by straightforward enumeration. Let $U \subset \mathcal{N}$ be a set of three customers. The corresponding inequality is defined as $\sum_{l\in\mathcal{L}} \sum_{r\in\Omega_l} \left\lfloor \frac{g_r}{2} \right\rfloor \lambda_r \leq 1$, where g_r is the number of times route r visits customers in U. Denote by $\sigma \leq 0$ its associated dual price. For every second visit to a customer in U, σ has to be subtracted from the reduced cost of a route. This complicates the solution of the pricing subproblem (see Section 3.3).

Recently, Pecin et al. (2017) proposed the use of limited memory SR (lmSR) which are a generalization of SR and whose impact on the solution of the pricing subproblem is typically reduced compared to standard SR. With each lmSR are associated a set of (three) customers U and a memory set M of nodes with $U \subseteq M \subseteq \mathcal{N}$. The basic idea of lmSR is similar to the ng-routes. Roughly speaking, each time a route r visits a node $j \notin M$ not in the memory of the cut, the coefficient g_r 'forgets' a previous visit to one of the customers $i \in U$ if the 'remembered' number of visits up to node j is odd. In the following section, we clarify why this simplifies the solution of the pricing subproblem if $|M| < |\mathcal{N}|$, and we give details on the computation of the coefficient g_r. For a detailed description of lmSR, the computation of the coefficient g_r, and the determination of the smallest-possible memory sets M we refer to Pecin et al. (2017).

Dynamic neighborhood extension As mentioned in Section 3.1, the quality of the lower bounds depends on the choices of the neighborhoods N_i of the ng-route relaxation. However, it is not clear a priori what good choices for N_i are. Roberti and Mingozzi (2014) proposed the dynamic extension of these neighborhoods, which can be interpreted as adding valid inequalities to formulation (1) forbidding routes with certain cycles. Let r be a route of the current LP solution that contains a cycle $C = (i, \ldots, i)$ with $i \in \mathcal{N}$. Then, we add customer i to the neighborhoods N_j of all nodes $j \in C$, forbidding this cycle in all routes that are priced out. In addition, all routes that are not feasible with respect to the new neighborhoods are removed from the master program (1).

3.3 Labeling algorithm

The predominant technique to solve ESPPRCs are dynamic-programming labeling algorithms (Irnich and Desaulniers 2005). In labeling algorithms, partial paths are gradually extended in a network looking for a minimum-cost path from a given source node to a given sink node. The partial paths are represented by labels storing the accumulated cost and resource consumption along the partial path. To avoid a complete enumeration of all feasible paths, dominance relations between different labels as well as other fathoming rules are typically exploited to eliminate unpromising labels. For a more comprehensive discussion on ESPPRCs and labeling algorithms, we refer to Irnich and Desaulniers (2005).

Forward labeling In the VRPPC, source and sink node of the pricing network are both given by the depot v_0. A partial path $P = (v_0, \dots, i)$ from the depot v_0 to a vertex $i \in \mathcal{V}$ is represented by a label $L(P) = (\tilde{c}(P), v(P) = i, q(P), \Pi(P), S(P))$ storing its reduced cost $\tilde{c}(P)$, its last vertex $v(P)$, the load $q(P)$ of the vehicle, the set $\Pi(P)$ of visited customer nodes (in the ng-sense), and a binary vector $S(P)$ representing the states of the lmSR. Let Θ be the set of all lmSR with strictly negative dual price in the current pricing iteration. We denote by $S_s(P)$, U_s, σ_s, and M_s the state, customer set, dual price, and memory associated with a lmSR $s \in \Theta$. The initial label at the depot v_0 is given by $(0, v_0, 0, \emptyset, \mathbf{0})$. The extension of a label $L(P)$ to a node $j \in \mathcal{V}$ along edge $\{v(P), j\} \in \mathcal{E}$ is feasible if $q(P) + q_j \leq C_l$ and $j \notin \Pi(P)$. If the extension is feasible, a new label $L(P') = (\tilde{c}(P'), j, q(P'), \Pi(P'), S(P'))$ is created according to the following resource extension functions (REFs):

$$\tilde{c}(P') = \tilde{c}(P) + \tilde{c}^l_{v(P)j} - \sum_{s \in \Theta: j \in U_s \wedge S_s(P) = 1} \sigma_s \tag{4}$$

$$v(P') = j \tag{5}$$

$$q(P') = q(P) + q_j \tag{6}$$

$$\Pi(P') = (\Pi(P) \cup \{j\}) \cap N_j \tag{7}$$

$$S_s(P') = \begin{cases} 0 & \text{if } j \notin M_s \vee (j \in U_s \wedge S_s(P) = 1) \\ 1 & j \in U_s \wedge S_s(P) = 0 \\ S_s(P) & otherwise \end{cases} \qquad \forall s \in \Theta \tag{8}$$

To eliminate unpromising labels that cannot lead to an improved complete path compared to another label, the following dominance rule is used. A label $L(P_1)$ dominates another label $L(P_2)$ with the same last vertex i if

$$\tilde{c}(P_1) - \sum_{s \in \Theta: S_s(P_1) > S_s(P_2)} \sigma_s \leq \tilde{c}(P_2), \tag{9}$$

$$q(P_1) \leq q(P_2), \tag{10}$$

$$\Pi(P_1) \subseteq \Pi(P_2). \tag{11}$$

REFs (4) and (8) together with the dominance relation (9) give the intuition of how the lmSR are handled in the labeling algorithm and why their impact on the solution of the pricing subproblems is reduced compared to the standard SR. The overall handling of the lmSR is analog to the SR. For every second visit to a customer $i \in U_s$ (there can be several visits to the same customer in an ng-route), the dual price σ_s has to be incorporated in the reduced cost. Thus, a binary representation of the state is sufficient for each cut $s \in \Theta$, and the state $S_s(P)$ changes whenever a node $i \in U_s$ is visited. Additionally, the state of cut s is reset in the lmSR case whenever a customer is visited that is not in the memory set M_s. In the dominance rule, two labels $L(P_1)$ and $L(P_2)$ with different states $S_s(P_1)$ and $S_s(P_2)$ for cut s can still be compared by penalizing the dominating label $L(P_1)$ if it is inferior with respect to the state of s (Jepsen et al. 2008). When using lmSR instead of SR, many more labels are directly comparable without penalization because the states for all lmSR $s \in \Theta$ for which $v(P) \notin M_s$ are reset.

In the remainder of this section, we describe several techniques that are used to speed-up the pricing process, namely bidirectional labeling, completion bounds, edge elimination, and heuristic pricing.

Bidirectional labeling In labeling algorithms, the number of generated labels typically increases strongly with the length of the generated partial paths. Bounded bidirectional labeling, originally introduced by Righini and Salani (2006) and successfully used in many state-of-the-art approaches to VRP variants, can help mitigate this effect and is therefore typically superior to its monodirectional counterpart. In bidirectional labeling, forward and backward partial paths are extended only up to a so-called halfway point (HWP) defined on one of the resources (that needs to be monotone). After the labeling process, suitable forward and backward partial paths have to be merged to complete feasible paths. Recently, Pecin et al. (2017) and Tilk et al. (2017) proposed the use of a HWP that is dynamically detected during the bidirectional labeling process based on the expected remaining forward and backward work. The idea is to reduce the overall workload by better balancing the necessary forward and backward labeling because they might be unequally complex due to asymmetry in the instance data or in the labeling itself. The VRPPC pricing subproblem instances of this section that use the reduced cost $\tilde{c}^l_{ij}, l \in \mathcal{L}$ from (2), however, are completely symmetric so that forward and backward labeling are essentially identical. As a consequence, it is sufficient to perform only the forward labeling up to the HWP. The resulting labels can then be interpreted as both forward and backward partial paths.

The bidirectional labeling algorithm for the VRPPC pricing subproblems works as follows. The HWP is defined on the load resource $q(P)$ of the labels. Forward labeling is then executed, extending a label $L(P)$ only if $q(P) \leq C_l/2$ holds. When the labeling process terminates, suitable labels $L(P)$ and $L(P')$ are merged. To avoid creating the same path from different combinations of labels, the 'first' label $L(P)$ is a candidate for merging only if $q(P) > C_l/2$ or $v(P) = v_0$. The two labels $L(P)$ and $L(P')$ can be merged to a complete feasible route if they end at the same vertex $(v(P) = v(P'))$, the capacity of the vehicle is not exceeded $(q(P) + q(P') - q_{v(P)} \leq C_l)$, and the sequence of customer visits is feasible in the ng-route sense $(|\Pi(P) \cap \Pi(P')| = 1)$. The reduced cost of the resulting route are

$$\tilde{c}(P) + \tilde{c}(P') - \sum_{s \in \Theta : v(P) \notin U_s \wedge S_s(P) + S_s(P') = 2} \sigma_s + \sum_{s \in \Theta : v(P) \in U_s \wedge S_s(P) + S_s(P') = 0} \sigma_s.$$

Completion bounds The dominance between labels allows the elimination of unpromising labels if there exists a label that is provably superior. Another strategy to discard unpromising labels is the use of completion bounds. The basic idea is to compute lower bounds for the cost of completing a label $L(P)$ to a feasible route. Clearly, if the resulting estimated reduced cost of a complete route is not negative, the corresponding label can be discarded because the real reduced cost of the route cannot be negative. Completion bounds are typically obtained by running the labeling algorithm on a relaxation of the pricing subproblem in the opposite direction, i.e., solve the relaxed problem with backward labeling to obtain completion bounds for the original problem solved with a forward labeling algorithm. As pointed out before, labeling in the VRPPC pricing subproblem is completely symmetric so that completion bounds for the forward labeling of the bidirectional labeling algorithm can be obtained by using the forward labeling on a relaxed version of the problem. In our approach, we relax the pricing subproblem in two different ways. First, we use smaller ng-neighborhoods N_i. Second, we consider only a fraction of the lmSR cuts (those with smallest dual price) explicitly in the labeling. As proposed by Contardo and Martinelli (2014), the effect of the remaining lmSR is partly incorporated into the completion bounds by subtracting $\sigma_s/2$ from the reduced cost $\tilde{c}^l_{ij}, l \in \mathcal{L}$ of the edges $\{i, j\}$ with $i, j \in U_s$.

Let $\hat{c}(i, q)$ be the minimum reduced cost of a label at vertex i with load q that results from the forward labeling algorithm solving the described relaxation of the pricing subproblem for vehicle type $l \in \mathcal{L}$. Then, in the bidirectional labeling algorithm for the original pricing subproblem of l, all labels $L(P)$ at node $v(P)$ for which $\tilde{c}(P) + \min_{q \le C_l - q(P)} \hat{c}(i, q) \ge 0$ holds are discarded.

Edge elimination A final acceleration technique that we use in our algorithm is the elimination of edges that cannot be part of any optimal solution as proposed by Irnich et al. (2010). Define the edge reduced cost $\hat{c}^l_{ij}$ for edge $\{i, j\} \in \mathcal{E}$ and vehicle type $l \in \mathcal{L}$ as the minimal reduced cost of any route $r \in \Omega_l$ regarding the current dual solution that uses edge $\{i, j\}$. If for some $\{i, j\} \in \mathcal{E}, l \in \mathcal{L}$ the edge reduced cost $\hat{c}^l_{ij}$ are larger than the current integrality gap, then edge $\{i, j\}$ can be removed from the pricing subproblem for vehicle type $l \in \mathcal{L}$. Irnich et al. (2010) have shown that the values $\hat{c}^l_{ij}$ for all edges $\{i, j\} \in \mathcal{E}$ can be computed by concatenating forward and backward labels $L(P)$ with $v(P) = i$ and $L(P')$ with $v(P') = j$ resulting from a call to the full forward and backward labeling algorithm for $l \in \mathcal{L}$. Again, due to symmetry reasons, labels from the forward labeling can be interpreted also as backward labels in the VRPPC so that a single call to the forward algorithm is sufficient to compute the $\hat{c}^l_{ij}$ for each $l \in \mathcal{L}$.

Heuristic pricers For the column-generation process, it is not necessary to identify a route with minimal reduced cost in every iteration. Instead, it is sufficient to provide any route with negative reduced cost. Consequently, pricing heuristics can be used to solve the pricing subproblems as long as they find such routes. The exact solution algorithm for the pricing subproblems only has to be invoked if the heuristic pricers fail to identify additional routes. In our BPC approach, we use limited discrepancy search (LDS, Feillet et al. 2007) to solve the pricing subproblems heuristically. The basic idea of LDS is to divide the set of edges into good and bad edges and to limit the number of bad edges that are allowed in the routes by discarding labels that exceed the allowed number. We define for each node the five best edges (w.r.t. reduced cost) as good edges, all other edges are bad edges. Furthermore, we consider two different values (zero and one) for the number of allowed bad edges, giving rise to two different heuristic pricers with differing computational effort. The pricing solvers are then executed in the following order: LDS with no bad edges, LDS with one bad edge, the exact labeling algorithm. Within the three pricing algorithms, the pricing subproblems for the different vehicle types $l \in \mathcal{L}$ are solved in the order of increasing capacity C_l. Whenever one or more routes with negative reduced cost are found by a combination of pricing solvers and pricing subproblem, they are returned to the master program, and the remaining solver-subproblem combinations are not invoked in this pricing iteration.

3.4 Branching

Denote by $\bar{\lambda}_r$ and $\bar{y}_i$ the values of variables λ_r and y_i in a solution of the LP-relaxation of model (1). Furthermore, let $\bar{x}_{ij} = \sum_{l \in \mathcal{L}} \sum_{r \in \Omega_l} b_{ijr} \bar{\lambda}_r$ be the aggregated flow over edge $\{i, j\} \in \mathcal{E}$ in this solution. Recall that b_{ijr} gives the number of times edge $\{i, j\}$ is traversed by route r.

We use the following hierarchical branching scheme. First, we branch on the overall number of customers served by the common carrier $\sum_{i\in\mathcal{N}} \bar{y}_i$. Second, we branch on the number of vehicles of type $l \in \mathcal{L}$ given by $\sum_{r\in\Omega_l} \bar{\lambda}_r$. If this is fractional for several vehicle types, we branch on the one that is closest to 0.5. Third, we branch on single y_i variables and the customer with $\bar{y}_i$ closest to 0.65 is chosen first. Finally, we branch on the edges $\mathcal{E}$ of the undirected graph $\mathcal{G}$ giving priority to the edge $\{i, j\}$ for which $\bar{x}_{ij}$ is closest to 0.5.

All branching decision can be implemented by adding a single constraint to model (1). Moreover, the pricing subproblems remain structurally unchanged, and all branching decision preserve the inherent symmetry of the VRPPC. The node selection strategy is best first.

4 Large neighborhood search for the VRPPC

This section describes our LNS for solving VRPPC. LNS was introduced by Shaw (1998) as a local search method with larger moves that make distant solutions accessible. The large moves are realized by means of a removal and an insertion step: a possibly large part of the solution, i.e., in the context of VRPs a subset of customers, is removed and then reintegrated into the partial solution. A similar approach was proposed by Schrimpf et al. (2000) as ruin and recreate. Ropke and Pisinger (2006b) introduced adaptive LNS, which allows to use a range of different operators that are selected with a probability depending on their past success. Our LNS is based on the latter approach, but we select each operator with the same fixed probability instead of adapting probabilities during the search. Figure 1 shows our solution method in pseudocode.

```
$\mathcal{S}_c \leftarrow$ generateInitialSolution()
**for** $\eta$ iterations **do**
    $\mathcal{S}_t \leftarrow \mathcal{S}_c$
    **for** two rounds **do**
        **for all** $r \in \mathcal{S}_t$ **do**
            {Decompose solution into subproblem $\mathcal{P}_r$ for selected route $r$. Generate solution of subproblem
            from the tentative solution of the original problem.}
            $\mathcal{S}_{\mathcal{P}_r} \leftarrow$ copyPartialSolution($\mathcal{P}_r, \mathcal{S}_t$)
            {Randomly select removal operator. Draw $\delta$ customers to remove from $\mathcal{S}_{\mathcal{P}_r}$.}
            $\mathcal{S}_{\mathcal{P}_r} \leftarrow$ applyRemoval($\mathcal{S}_{\mathcal{P}_r}, \delta$)
            {Randomly select insertion operator to reinsert customers.}
            $\mathcal{S}_{\mathcal{P}_r} \leftarrow$ applyInsertion($\mathcal{S}_{\mathcal{P}_r}$)
            **if** acceptSA($\mathcal{S}_{\mathcal{P}_r}, \mathcal{S}_t$) **then**
                $\mathcal{S}_t \leftarrow$ integrate($\mathcal{S}_{\mathcal{P}_r}, \mathcal{S}_t$)
            **end if**
        **end for**
    **end for**
    **if** $\mathcal{S}_t$ is feasible and with a probability of $0.25$ **then**
        $\mathcal{S}_t \leftarrow$ VND($\mathcal{S}_t$)
    **end if**
    updatePenaltyFactor($\mathcal{S}_t$)
    $\Omega^{LNS} \leftarrow$ addFeasibleRoutes($\mathcal{S}_t$)
    **if** acceptSA($\mathcal{S}_t, \mathcal{S}_c$) **then**
        $\mathcal{S}_c \leftarrow \mathcal{S}_t$
    **end if**
**end for**
$\mathcal{S}_{best} \leftarrow$ postProcessing($\Omega^{LNS}$)
```

Figure 1: Overview of the LNS algorithm.

First, we generate an initial feasible solution $\mathcal{S}_c$ with a modified savings algorithm (Section 4.1). In the following improvement phase, we allow infeasible solutions and penalize violations in the objective function (Section 4.2). The improvement phase (Section 4.3) works as follows: In every iteration, we decompose the original problem into a sequence of subproblems $\mathcal{P}_r$. Then, we derive a first solution $S_{\mathcal{P}_r}$ to subproblem $\mathcal{P}_r$ from the tentative solution $\mathcal{S}_t$ and apply a randomly selected removal and insertion operator on the solution of the subproblem. Afterwards, the new solution is reintegrated into the current solution of the original problem based on a simulated annealing (SA) acceptance criterion. After all subproblems are processed, we repeat the procedure for another round. If the resulting complete solution is feasible, we apply a variable neighborhood descent (VND) with a probability of 0.25. The final acceptance decision of the iteration is again based on SA. Finally, we save every feasible route found during the search. In a post-processing step, these routes are recombined into the best possible feasible solution (Section 4.4).

4.1 Generation of initial solution

We use a modified savings algorithm (see Clarke and Wright 1964) to create an initial feasible solution for the VRPPC. The idea is to first serve each customer with a dedicated route, and then to merge pairs of routes as long as a positive saving can be realized and vehicle capacity is not violated. In each iteration, we merge the pair with the highest saving. To merge two routes r_1 and r_2, we consider only the edges incident to the depot and remove one edge of r_1 and one edge of r_2. Then, we replace them by an edge directly linking the corresponding customer i of r_1 and j of r_2.

In the basic version of the algorithm, a tendency to favor peripheral routes can be observed, i.e., to prefer routes that serve customers that are far from the depot and to eventually isolate customers that are located close to the depot (Gaskell 1967). We calculate the saving $s(i,j)$ when linking customers i and j as $s(i,j) = c_{0i} + c_{j0} - \lambda \cdot c_{ij}$, where λ is a weight to balance between the distance to the depot and the distance between customers (see, e.g., Gaskell 1967, Yellow 1970). We randomly select $\lambda \in [0.6, 1.6]$ using the values proposed in Li et al. (2005) to define the interval. To ensure that at most $|\mathcal{K}|$ vehicles are used, we implement the following simple procedure. In the beginning, none of the initial single-customer routes are assigned to vehicles. We only merge two single-customer routes if an unused vehicle is available, in this case we assign a vehicle to the resulting two-customer route. Otherwise, we refrain from merging the two routes. When two multiple-customer routes are merged, we release one vehicle. Finally, all the customers of the remaining single-customer routes are assigned to the common carrier.

4.2 Generalized cost function and penalty calculation

During the search, we allow solutions that violate the vehicle capacity constraint and add a penalty to the objective function value to account for the respective violation. Although restoring feasibility of a solution is always possible by assigning certain customers to the common carrier, we allow temporary violations to be able to traverse the solution space more freely. The objective value of a solution $\mathcal{S}$ is given by the generalized cost function $f_{gen}(\mathcal{S})$ (see, e.g., Gendreau et al. 1994):

$$f_{gen}(\mathcal{S}) = f(\mathcal{S}) + \gamma \cdot L(\mathcal{S}) = f_{var}(\mathcal{S}) + f_{\mathit{fix}}(\mathcal{S}) + \gamma \cdot L(\mathcal{S}).$$

The term $f(S)$ comprises two parts: $f_{var}(S)$ denotes the cost for the distance traveled by the private fleet and the cost of outsourcing customers to the common carrier; $f_{\mathit{fix}}(\mathcal{S})$ denotes the cost of using the vehicles of the private fleet. The penalty for capacity violations is calculated as the product of a penalty factor γ and the total capacity violation $L(\mathcal{S})$ of solution S. For all operators that our algorithm uses to modify a solution, $L(\mathcal{S})$ can be calculated in $\mathcal{O}(1)$ time.

We initially set the penalty factor to $\gamma = \bar{h}_{max}$, where $\bar{h}_{max}$ denotes the maximum cost per demand unit to subcontract any customer, i.e., $\bar{h}_{max} = \max_{i \in \mathcal{N}} (h_i/q_i)$. Then, we update γ as follows: If the capacity constraint has been violated for two iterations, the penalty factor is multiplied by ϱ, and vice versa it is divided by ϱ if the capacity constraint is satisfied for two iterations. We restrict the value of the penalty factor to the interval $\gamma \in [0.001, 10.0 \cdot \bar{h}_{max}]$.

4.3 Solution improvement

In every iteration of the improvement phase, we decompose the original problem into a series of subproblems $\mathcal{P}_r$, one for each route $r \in \mathcal{S}_t$, and solve these subproblems one at a time. However, instead of solving each subproblem from scratch, we derive a solution $\mathcal{S}_{\mathcal{P}_r}$ of $\mathcal{P}_r$ from $\mathcal{S}_t$ and improve this solution (Section 4.3.1). Improvement is achieved by applying a randomly selected removal and insertion operator to $\mathcal{S}_{\mathcal{P}_r}$ (Section 4.3.2). Then, an acceptance criterion based on SA decides whether $\mathcal{S}_{\mathcal{P}_r}$ is integrated into $\mathcal{S}_t$ or discarded (Section 4.3.3). If we accept the new solution, we immediately replace the corresponding routes in $\mathcal{S}_t$ with the modified routes and update the customers assigned to the common carrier according to $\mathcal{S}_{\mathcal{P}_r}$. Note that the definition of the next subproblem to be investigated depends on the solution of the current subproblem because we modify $\mathcal{S}_t$ continuously. The entire procedure is repeated for two rounds (in every round one subproblem $\mathcal{P}_r$ originates from each route r).

If the resulting solution is feasible, we further improve the solution by applying a VND with a probability of 0.25 (Section 4.3.4). Then, at the end of every iteration, the SA criterion (Section 4.3.3) decides whether the search is continued from the new solution. The search terminates after η iterations.

4.3.1 Decomposition strategy

Although problem decomposition is generally used in algorithms developed to tackle large-scale VRP instances (see, e.g., Vidal et al. 2013), we observed that even for medium-sized instances of the VRPPC, we often find better solutions if we focus the search on partial solutions instead of the overall problem. We define subproblems by means of the set of customers to be served and the vehicles available to carry out the service. Our decomposition strategy works as follows: In randomized order, we generate one subproblem $\mathcal{P}_r$ for each route r of the private fleet. Based on the tentative solution $\mathcal{S}_t$, each subproblem $\mathcal{P}_r$ is defined by:

i) *The customers and the vehicle that are currently assigned to route r.*

ii) *All customers and vehicles of a random number α of routes that are closest to route r.* The number of routes α is drawn from the interval $[2, \lfloor |\mathcal{K}|/4 \rfloor]$ if $|\mathcal{K}| \geq 12$ and set to $\alpha = \min(|\mathcal{K}| - 1, 2)$, otherwise. We select the α routes that are closest to route r by measuring the distance between routes as the Euclidean distance between their centers of gravity, where the center of gravity of a route is calculated as the average of the coordinates of the vertices of the route.

iii) *Customers that are closely-located to route r and that are currently assigned to the common carrier.* To select such customers, we create a rectangular box that contains all customers already present in $\mathcal{P}_r$ after steps i) and ii), and we add a margin to every side of the bounding box that corresponds to 10% of the maximal distance between any two customers in the instance. Then, we add all customers to $\mathcal{P}_r$ that are assigned to the common carrier in $\mathcal{S}_t$ and that are positioned within the enlarged box.

4.3.2 Removal and insertion operators

We remove δ customers from $\mathcal{S}_{\mathcal{P}_r}$. Let $|\mathcal{P}_r|$ denote the number of vertices in the current subproblem of our instance $\mathcal{P}_r$. Then δ is drawn from the interval $[\omega_{min}, \omega_{max}] \cdot \min(|\mathcal{P}_r|, 100)$ with parameters ω_{min} and ω_{max}.

Our LNS uses the following removal operators:

Random removal removes arbitrary customers from the routes of the private fleet and from the common carrier.

Route and common carrier removal selects at random routes from the private fleet or it selects the common carrier and removes the customers assigned until δ customers are removed.

Worst removal was proposed in Ropke and Pisinger (2006a) in order to remove customers whose presence in the solution strongly contributes to the objective function value. We do not use the direct contribution to $f_{gen}(\mathcal{S})$ to identify these customers but introduce problem-specific measures to select customers for removal. Whenever the worst removal operator is selected, we select one of three measures at random that is then used until δ customers are removed. Let S^{-i} denote a solution where customer i is removed and is either replaced by the connection between its predecessor and successor if it was previously assigned to the private fleet or removed without additional modification if it was assigned to the common carrier. For each customer i, let $W_x(i)$ denote the value of one of the following measures x:

i) change of variable cost: $W_1(i) = f_{var}(\mathcal{S}) - f_{var}(\mathcal{S}^{-i})$,

ii) change of variable cost compared to subcontracting: $W_2(i) = f_{var}(\mathcal{S}) - f_{var}(\mathcal{S}^{-i}) - h_i$,

iii) and change of variable cost per unit of demand: $W_3(i) = (f_{var}(\mathcal{S}) - f_{var}(\mathcal{S}^{-i}))/q_i$.

Measure W_1 is implemented in two variants, one variant including customers assigned to the common carrier and one variant ignoring these customers, and measure W_2 does not consider customers assigned to the common carrier. In the next step, the customers are sorted in descending order of $W_x(i)$, and the vertex at position $\lfloor D \cdot b^{\chi_{rem}} \rfloor$ is chosen, where D is the size of the list, b is a uniform random number $\in [0, 1]$, and χ_{rem} is a parameter to control the amount of diversification. After a customer is removed, the values are updated.

Shaw removal was introduced in Shaw (1997) in order to select customers for removal that are similar to each other. We define the similarity $R(i, j)$ between two customers i and j by their geographical distance d_{ij}, the difference in demand $|q_i - q_j|$, and the difference in subcontracting cost $|h_i - h_j|$. Each term is weighted with a parameter χ and normalized using the maximum value in the instance:

$$R(i,j) = \chi_d \frac{d_{ij}}{\max\limits_{i,j \in V}(d_{ij})} + \chi_q \frac{|q_i - q_j|}{\max\limits_{i \in \mathcal{N}}(q_i) - \min\limits_{i \in \mathcal{N}}(q_i)} + \chi_h \frac{|h_i - h_j|}{\max\limits_{i \in \mathcal{N}}(h_i) - \min\limits_{i \in \mathcal{N}}(h_i)}.$$

The first customer is randomly selected, and we sort all remaining customers $j \in \mathcal{S}$ in ascending order of their $R(i, j)$-value. From this list, the customer at position $\lfloor D \cdot b^{\chi_{rem}} \rfloor$ is chosen as described above. The next iteration starts from an already removed customer that is selected at random.

Historical node-pair removal was introduced in Ropke and Pisinger (2006b). The idea is to remove edges from the current solution that have so far been only present in solutions with poor quality. To this end, an auxiliary graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ is created and initially a large weight $w_{\tilde{e}}$ is assigned to each edge $\tilde{e} \in \tilde{\mathcal{E}}$. In the following, for each edge e that is present in the current solution S, the edge weight in the auxiliary graph is replaced if the current objective function value is smaller than the previous weight, i.e., $w_{\tilde{e}} := f(S)$ if $f(S) < w_{\tilde{e}}$.

To select customers, we assign to every customer served by the private fleet the sum of the weights $w_{\tilde{e}}$ of the two edges incident to the customer and then sort the customers in descending order according to this value. Now, the customers with the highest value are removed. We adapt this procedure to also account for customers that are assigned to the common carrier (for which there are no edges) as follows: i) in addition to the edge weights, we store for each vertex $\tilde{v}$ a weight $w_{\tilde{v}}$ that contains the best objective function value of any solution encountered so far in which the customer was subcontracted and include them in the list, ii) because each customer served by the private fleet is assigned the sum of two edge weights, but customers served by the common carrier are only assigned one vertex weight, we divide each edge-related value by 2 before we add it to the list to make the measures comparable.

The following insertion operators are used in our LNS:

Greedy insertion basic determines for each unassigned customer i the minimal increase of the generalized cost function $\Delta f_{gen}(S^{+i})$ when i is inserted into the routes of the private fleet. Then, the customer with the smallest value of $\min(\Delta f_{gen}(S^{+i}), h_i)$ is assigned to either the best route of the private fleet at its best position (if the first term is smaller) or to the common carrier (if the second term is smaller). This is repeated until all customers are assigned.

Greedy insertion priority works similar to the basic version, but we modify the selection criterion for the next customer to insert. The idea is to prefer customers in the beginning that are expensive to subcontract and that have a low demand. Therefore, instead of selecting the customer i based on the minimum cost increase $\Delta f_{gen}(S^{+i})$, we calculate for each customer i) the difference between the minimum cost change for the assignment to the private fleet and the cost of subcontracting the customer in relation to its demand, i.e., $(\Delta f_{gen}(S^{+i}) - h_i)/q_i$, and ii) the direct subcontracting cost per unit, i.e., h_i/q_i. Then, we select the customer for insertion where the minimum of these two values is smallest. If the first value is smaller, we insert the customer at the cost-minimal position in the corresponding route of the private fleet. If the second value is smaller, we assign it to the common carrier.

Regret-2 insertion aims at finding a customer insertion order that tries to avoid negative future consequences, i.e., we insert a customer now because otherwise we might regret it. A description of the regret-k insertion is given in Ropke and Pisinger (2006a), we only implement the case $k = 2$ because larger values of k did not improve the solution quality in preliminary studies. The regret-2 value is calculated as the difference between the second best assignment to a route of the private fleet or to the common carrier and the best assignment. In every step, we choose the customer with the currently highest regret-2 value and insert this customer into the best route or assign it to the common carrier, whichever is cheaper. If the best assignment corresponds to the common carrier, we assume a regret-2 value of zero because the common carrier has unlimited capacity.

Insertion diversification adapts the three previously introduced insertion operators by adding an additional term to the change of the objective function value. On the one hand, we set the cost h_i of subcontracting customer i to $h_i := h_i + \iota$ where ι is uniformly chosen from the interval $\iota \in [-\zeta \cdot h_{max}, \zeta \cdot h_{max}]$ with $h_{max} = \max_{i \in \mathcal{N}} (h_i)$ and parameter ζ in order to try different configurations of subcontracted customers. On the other hand, for assignments to the private fleet, we use a principle known as continuous diversification (Cordeau et al. 2001) that originated in the context of tabu search but is less restrictive than using a tabu list. Assignments to routes of the private fleet are changed based on the history of the solution process in order to encourage customer-route combinations that have not occurred very frequently. To this end, we measure the frequency u_{ij} of assigning customer i to route j and derive a penalty term $\kappa \cdot |\mathcal{K}| \cdot \sqrt{u_{ij} \cdot f(\mathcal{S}_{best})/|\mathcal{N}|}$ that grows sublinearly with this frequency. The penalty depends on a parameter $\kappa = 0.1$ that controls the extent of diversification, the number of vehicles $|\mathcal{K}|$, and the currently best objective function value per customer $f(\mathcal{S}_{best})/|\mathcal{N}|$.

Probabilistic insertion randomizes the order of customer insertions and the decision whether to assign a customer to one of the vehicle routes or to the common carrier. The next customer i is selected randomly, and then we calculate the changes of the objective function value $\Delta f^x_{gen}(S^{+i})$ with $x = r$ for assigning i to route r at its cost-minimal position and with $x = c$ for assigning i to the common carrier. Now, we use roulette wheel selection with probabilities inversely proportional to the changes in the objective value to decide whether to assign customer i to route r at the best position on the route or to the common carrier, i.e., the higher the increase when assigned to x, the lower the probability to select it. The parameter χ_{ins} controls the amount of diversification.

4.3.3 Acceptance criterion

We use an SA-based acceptance criterion (Kirkpatrick et al. 1983) to decide i) whether to replace the solution to each subproblem with the newly generated solution for the subproblem, and ii) whether to continue the search from the tentative solution $\mathcal{S}_t$ after all subproblems have been solved or to continue from the current solution $\mathcal{S}_c$. Based on the temperature T and the difference between the objective function value of a new solution $\mathcal{S}_{new}$ and the previous solution $\mathcal{S}_{old}$, SA decides whether to accept deteriorating solutions (improving ones are always accepted) with probability $p(\mathcal{S}_{new}, \mathcal{S}_{old}, T) = e^{(f_{gen}(\mathcal{S}_{old}) - f_{gen}(\mathcal{S}_{new}))/T}$.

We set start and end temperature such that a new solution that deteriorates the initial solution by $\tau_{max}\%$ and $\tau_{min}\%$, respectively, is accepted with a probability of 50%. After each iteration, T decreases by a constant factor, which is determined such that the end temperature is reached after 50% of the total iterations. From there, we keep T constant instead of further decreasing it to allow more diversification in the search. Pretests have shown that this has strong positive effects on the solution quality. To evaluate the acceptance of a new solution $\mathcal{S}_{\mathcal{P}_r}$, we determine $f_{gen}(\mathcal{S}_{new})$ as the objective value of the complete solution that would be obtained if $\mathcal{S}_{\mathcal{P}_r}$ replaced the previous solution to $\mathcal{P}_r$. This is necessary because the temperature T is scaled to the value of complete solutions of the original problem.

4.3.4 Variable neighborhood descent

After two rounds of problem decomposition and improvement using LNS, feasible solutions are improved by a VND with a probability of 0.25. The VND follows a first-improvement strategy, and the list of neighborhoods contains the following operators in the given order: relocate (Waters 1987), exchange (Savelsbergh 1992), and a restricted version of 2-add-drop (Bolduc et al. 2008). Relocate and exchange are implemented in inter- and intra-route fashion. The 2-add-drop operator is specific to the VRPPC and was originally introduced as combined operator that i) transfers up to two customers from the common carrier to the private fleet or vice versa, and ii) transfers a single customer from the private fleet to the common carrier and at the same time inserts another customer currently assigned to the common carrier at the best possible position within the routes of the private fleet. We use only the second variant.

To limit the computational effort, we restrict the search to promising moves as follows: For each customer i, we store the closest $0.3 \cdot \min(|\mathcal{V}|, 150)$ vertices in an immutable neighbor list. We generate only those relocate moves of which either the new successor or predecessor of i is contained in the neighbor list of i, but we always evaluate the move where i is assigned to the common carrier. For exchange moves, only vertices in the neighbor list of i are considered as exchange partners of i. For 2-add-drop moves, we do not restrict the search to closely located vertices because good insertion positions can be far from the removal position.

4.4 Set covering with fleet constraints

Finally, we apply a post-processing step that aims at improving on the best found solution by solving a set-covering problem with constraints on the fleet composition. Similar techniques have been successfully used in, e.g., Rochat and Taillard (1995), Groër et al. (2011), and Subramanian et al. (2013). For each vehicle group $l \in \mathcal{L}$, we collect all feasible routes encountered during the search in a pool of routes Ω_l^{LNS}. Whenever we find a route of a vehicle of group l that serves the same customers as a route already present in the respective pool but with lower cost, we replace the corresponding route. Then, we solve formulation (1) using a commercial solver with a time limit of $\min(300, |\bigcup_{l\in\mathcal{L}} \Omega_l^{LNS}|/100)$ seconds. We initialize the solver with $\mathcal{S}_{best}$ in order to decrease the computing time. If we obtain a solution in which customers are contained in more than one route, we simply remove the redundant occurrences.

5 Numerical studies

This section details the experiments to assess the performance of our BPC and our LNS. Section 5.1 introduces the benchmark instances on which both algorithms are evaluated. Section 5.2 discusses the parameter tuning, the value of individual components, and the comparison to the state-of-the-art for our LNS. Results of our BPC algorithm are presented in Section 5.3.

5.1 Test instances

In our computational studies, we use the benchmark sets available from the literature. Bolduc et al. (2008) introduce two sets that assume a homogeneous vehicle fleet. These sets are based on the well-known capacitated VRP instances of Christofides et al. (1979) (14 instances with 50–199 customers) and Golden et al. (1998) (20 instances with 200–483 customers). The names of the instances of these sets start with CE and G, respectively. To obtain VRPPC instances, the original instances are adapted as follows: i) the number of vehicles is reduced such that only 80% of the total demand of the customer can be satisfied by the private fleet, ii) the vehicle fixed costs are set based on the average route length of the best known solution to the original instance, and iii) the cost of subcontracting a customer is based on the best known objective value of the original instance, the distance of the customer to the depot, and its demand. To obtain instances with a heterogeneous fleet, both sets are further modified in Bolduc et al. (2008). The vehicles are divided into two or three different vehicle types, with 80%, 100% and 120% of the capacity and of the fixed cost of the vehicles of the homogeneous fleet. The instance names of these sets start with CE-H and G-H, respectively.

5.2 Performance of our LNS

Experimental environment and parameter setting For our LNS, we perform all numerical experiments with a single core of a desktop computer equipped with an Intel I7 processor at 2.8 GHz with 8 GB of RAM and running Windows 7 Enterprise. The algorithm is implemented in Java, and the commercial solver used to solve the set-covering problem (Section 4.4) is Gurobi at version 7.0.1. Ten runs per instance are performed.

We set the total number of search iterations to $\eta = 20,000$ because this value offers a good trade-off between run-time and solution quality. We tune the other parameters of the algorithm as follows: We use only three randomly selected instances from the set CE and three instances from the set G to keep the computational tuning effort low and to avoid overfitting the algorithm to the benchmark set. We begin with a reasonable base setting of the parameters that we have determined during the development of our algorithm. Then, we iteratively modify the base value of each parameter to a reasonable lower and higher value. We keep the best value for each parameter (based on the average quality of 10 runs) and continue with the next parameter. Parameters that are closely related are grouped and changed simultaneously to keep the testing effort moderate. We examine the following parameters in the given order: The external cost noise factor (ζ), the weight factors for the Shaw removal operator (χ_d, χ_h, χ_q), the minimal and maximal factors for the number of customers to remove $(\omega_{min}, \omega_{max})$, the removal diversification factor (χ_{rem}), the continuous diversification factor (κ), the probabilistic insertion factor χ_{ins}, the minimal and maximal SA deterioration percentage (τ_{min}, τ_{max}), and finally the penalty update factor ϱ. Table 1 summarizes the results of our parameter study. The base value is given in the middle and the best value is marked in bold and used as final setting. For each setting, we report the deviation in percent of the objective value Δ_f to the value of the best setting. We conclude that our solution method is quite robust against parameter variations as the deviation from the best setting is always below 0.2%.

LNS			
ζ	0.25	$\mathbf{0.5}$	0.75
$\Delta_f(\%)$	0.02	0.00	0.1
(χ_d, χ_h, χ_q)	$(\mathbf{4}, \mathbf{5}, \mathbf{6})$	$(6, 5, 4)$	$(6, 4, 5)$
$\Delta_f(\%)$	0.00	0.03	0.19
$(\omega_{min}, \omega_{max})$	$(0.05, 0.4)$	$(\mathbf{0.1}, \mathbf{0.4})$	$(0.1, 0.6)$
$\Delta_f(\%)$	0.09	0.00	0.02
χ_{rem}	26	**36**	46
$\Delta_f(\%)$	0.16	0.00	0.03
κ	$\mathbf{0.02}$	0.1	0.5
$\Delta_f(\%)$	0.00	0.04	0.17
χ_{ins}	2	**4**	6
$\Delta_f(\%)$	0.07	0.00	0.09
SA & Penalties			
τ_{min}	0.01	$\mathbf{0.05}$	0.1
$\Delta_f(\%)$	0.17	0.00	0.03
τ_{max}	0.1	$\mathbf{0.2}$	0.4
$\Delta_f(\%)$	0.07	0.00	0.02
ϱ	1.02	$\mathbf{1.1}$	1.5
$\Delta_f(\%)$	0.14	0.00	0.12

Table 1: Results of our parameter study on a subset of the VRPPC instances. We mark the best setting for each parameter in bold and use it as final setting. For each setting, we report the deviation (Δ_f) to the best setting of the respective parameter.

Influence of algorithmic components To assess the effect that different components of the algorithm have on the quality and speed of our LNS, we compare the following variants of the LNS on the benchmark sets with homogeneous (Table 2) and heterogeneous fleet (Table 3):

LNS Our LNS with all components as described above

LNS–noSC In this setting, we omit the set covering as post-processing step. To be able to observe the undistorted effect of the post processing, the results of LNS–noSC and LNS presented in the following are based on the same runs: LNS–noSC is the result of the LNS before the post-processing phase starts.

LNS–noDec This setting does not use the problem decomposition technique but instead applies the removal and insertion operators on complete solutions. To make the comparison fair, we double the number of total search iterations because we always do two rounds of destroy and repair in the variants featuring decomposition.

For each instance, column $|\mathcal{N}|$ reports the number of customers, column BKS the previously best known solution value from the literature (in the case of Table 2 taken from Vidal et al. (2016) and in the case of Table 3 taken from Bolduc et al. (2008), Potvin and Naud (2011) and from the updated results for Côté and Potvin (2009) reported in Appendix A). For each variant of the LNS, column Δ_{best} reports the percentage gap between the best solution obtained in the 10 runs and the previous best known solution, and column t gives the run-time in seconds. Finally, columns $\overline{\text{LNS}}$ list the best objective value f that we found during the overall testing and the respective gap to the BKS Δ_f. Note that the best solution value for each instance is marked in bold, and average values are provided in the last row.

The solution behavior is rather similar on both types of instances: The two variants using decomposition show a clearly superior solution quality while having higher run-times. Of these two, LNS shows a better solution quality than LNS–noSC, but the average run-time increases by roughly 25%. The results show that both algorithmic components—decomposition and post-processing—have a positive impact on solution quality and a negative one on run-time. In real-world applications, the decision for one of the variants depends on the desired tradeoff between solution quality and run-time that the planner wants to achieve. For the following comparison to the state-of-the-art, we put the major emphasis on solution quality and only investigate variant LNS.

Comparison to the state-of-the-art As Table 2 shows, LNS is able to match 10 and improve 6 previous BKS out of the 34 instances with a homogeneous vehicle fleet. During the overall testing, $\overline{\text{LNS}}$ matches 12 and improves 12 BKS of these instances. On the instances with a heterogeneous fleet (see Table 3), LNS matches 1 and improves 26 out of 34 instances, and $\overline{\text{LNS}}$ matches 3 and improves 29 instances.

Table 4 gives an aggregate comparison of LNS to the state-of-the-art methods from the literature: RIP (Bolduc et al. 2008), TS (Côté and Potvin 2009), TS+ (Potvin and Naud 2011), AVNS (Stenger et al. 2013b), AVNS-RN (Stenger et al. 2013a), UGHS (Vidal et al. 2016), MS-ILS (Vidal et al. 2016), and MS-LS (Vidal et al. 2016). The upper part of the table is devoted to the comparison on the instances with a homogeneous fleet, the lower part to those with a heterogeneous fleet. The first three rows in each part report the average percentage gap to the previous best known solution in percent achieved by each method in the best of the runs on the respective instance set. The solutions reported for RIP are obtained by solving each instance only once, all other heuristics report the best solution found in 10 runs. The fourth row gives the average run-time per instance in minutes. Finally, in row CPU@GHz, we list the processor and clock rate of the computers on which the respective methods were tested. We indicate with † results that are not directly comparable because they are obtained using truncated customer coordinates. Please note that the results reported for TS are based on non-truncated coordinates because the authors repeated the testing of their algorithm and provided us with this data. We report their updated results in Appendix A.

On the homogeneous-fleet instances, LNS provides the best solution quality on set CE and the second-best quality after UHGS on set G within very competitive run-times. On the heterogeneous-fleet instances, LNS is able to significantly improve the average solution quality by more than 1% compared to all competitors. However, run-times compared to TS are also clearly higher.

			LNS		**LNS–noSC**		**LNS–noDec**		$\overline{\textbf{LNS}}$	
Inst.	$\|\mathcal{N}\|$	BKS	$\Delta_{best}(\%)$	t(sec)	$\Delta_{best}(\%)$	t(sec)	$\Delta_{best}(\%)$	t(sec)	f	$\Delta_f(\%)$
CE										
CE-01	50	**1119.47**	**0.00**	27	**0.00**	27	**0.00**	13	**1119.47**	**0.00**
CE-02	75	**1814.52**	**0.00**	53	**0.00**	53	**0.00**	118	**1814.52**	**0.00**
CE-03	100	**1919.05**	**0.00**	73	**0.00**	73	0.11	67	**1919.05**	**0.00**
CE-04	150	**2505.39**	0.08	133	0.08	133	**0.00**	161	**2505.39**	**0.00**
CE-05	199	**3081.59**	0.23	298	0.25	272	0.35	374	3086.75	0.17
CE-06	50	**1207.47**	**0.00**	27	**0.00**	27	**0.00**	13	**1207.47**	**0.00**
CE-07	75	**2004.53**	**0.00**	53	**0.00**	53	**0.00**	109	**2004.53**	**0.00**
CE-08	100	**2052.05**	**0.00**	73	0.07	73	0.09	94	**2052.05**	**0.00**
CE-09	150	2422.74	**-0.12**	130	**-0.12**	128	-0.08	190	**2419.84**	**-0.12**
CE-10	199	3381.67	-0.12	288	-0.09	269	0.08	404	**3376.80**	**-0.14**
CE-11	120	**2330.94**	**0.00**	104	**0.00**	104	**0.00**	58	**2330.94**	**0.00**
CE-12	100	**1952.86**	0.04	73	0.04	73	**0.00**	82	**1952.86**	**0.00**
CE-13	120	**2858.83**	**0.00**	104	**0.00**	104	**0.00**	60	**2858.83**	**0.00**
CE-14	100	**2213.02**	**0.00**	72	**0.00**	72	**0.00**	65	**2213.02**	**0.00**
G										
G-01	240	**14131.18**	0.21	349	0.25	305	0.58	352	14134.20	0.02
G-02	320	**19142.75**	0.06	471	0.20	400	1.35	463	19145.60	0.01
G-03	400	**24409.02**	0.45	604	0.63	465	1.90	526	24671.36	1.07
G-04	480	34362.8	**-0.52**	731	-0.45	554	1.29	623	**34183.06**	**-0.52**
G-05	200	**14223.63**	1.07	193	1.19	189	0.70	132	14246.68	0.16
G-06	280	**21382.16**	0.92	279	1.01	264	0.54	216	21502.53	0.56
G-07	360	**23373.38**	0.50	550	0.51	399	1.21	464	23398.93	0.11
G-08	440	29797.62	**-0.34**	683	-0.10	510	1.59	587	**29697.75**	**-0.34**
G-09	255	1326.2637	0.35	460	0.35	323	0.29	437	**1325.03**	**-0.09**
G-10	323	1593.79492	0.13	697	0.13	498	0.43	474	**1586.50**	**-0.46**
G-11	399	2173.82151	0.30	917	0.30	627	0.63	510	**2163.72**	**-0.46**
G-12	483	2494.56071	0.27	1036	0.27	757	0.49	543	**2490.23**	**-0.17**
G-13	252	**2258.02**	0.08	869	0.08	661	0.99	536	2260.86	0.13
G-14	320	2683.73	0.00	1209	0.29	914	1.27	579	**2682.90**	**-0.03**
G-15	396	**3145.11**	0.07	1672	0.07	1370	0.74	637	3149.87	0.15
G-16	480	3620.71	0.11	2107	0.11	1803	0.82	695	**3614.79**	**-0.16**
G-17	240	**1666.31**	**0.00**	446	**0.00**	385	0.04	325	**1666.31**	**0.00**
G-18	300	**2730.55**	0.09	1015	0.17	712	0.39	560	2731.98	0.05
G-19	360	3497.2	-0.02	1476	0.06	1170	0.61	634	**3492.31**	**-0.14**
G-20	420	4312.45	-0.10	2080	-0.01	1771	0.90	734	**4303.56**	**-0.21**
Avg.			**0.11**	**569**	**0.16**	**457**	**0.51**	**348**		**-0.01**

Table 2: Detailed results and comparison of algorithmic components on the instance sets with a homogeneous fleet.

5.3 Results obtained with our branch-price-and-cut algorithm

All algorithmic components of the BPC were coded in C++ and compiled into 64-bit single-thread code with MS Visual Studio 2013. The callable library of CPLEX 12.6.0 was used for re-optimizing the master program. The computational experiments of the BPC were conducted on a standard PC with an Intel I7-5930k 3.5 GHz processor with 64 GB of main memory. The time limit was set to two hours.

Tables 5 and 6 summarize the results of our BPC on the homogeneous and heterogeneous instance sets, respectively. The columns have the following meaning: IUB is the initial upper bound given to the algorithm (from our LNS or from the literature), LB denotes the lower bound provided by the algorithm when the time limit was reached (values in bold indicate that the instance was solved to proven optimality within the time limit while values marked with an asterisk constitute new best known solutions), $\Delta_{LB}(\%)$ is the optimality gap in percent, t(sec) the run-time in seconds taken by the algorithm to solve an instance to proven optimality (or TL if the time limit was reached), #Nds is the number of solved branch-and-bound nodes, and #Cuts the number of generated cuts.

Our results reveal that the proposed exact approach is able to solve some small to medium-sized instances to optimality in reasonable time. Thereby, new best known solutions are found for two of the instances from benchmark set CE-H. For the larger instances, only lower bounds can be provided within the time limit of two hours. For the two largest instances G-12/G-H-12, we were not able solve the root node in the computation time of two hours. The average optimality gaps are 0.22%, 0.78%, 0.31%, and 1.10% for the instances sets CE, G, CE-H, and G-H, respectively. Thus, the lower bounds provided by our BPC seem to be rather tight while the upper bounds from our LNS (and from other state-of-the-art heuristics) also seem to be of good quality.

6 Conclusion

We present the first exact solution method for the VRPPC. Our BPC algorithm solves instances with up to 75 customers to optimality and provides tight lower bounds for instances with up to 480 customers. In addition, we propose an LNS as upper bounding procedure, which is among the best heuristic solution methods for the VRPPC. Our LNS features a decomposition procedure which may also be interesting for the solution of other VRP variants.

As future research, it seems worthwhile to extend the VRPPC to a planning problem spanning multiple days. This is motivated by applications in e-commerce, where often some customers have a desired delivery date, but others are indifferent about the concrete delivery date as long as it lies within a certain time frame. The idea here is to simultaneously solve the VRRPC on multiple days, i.e., one type of customers must be served on specified days, the other type can be scheduled freely, and those that cannot be served economically may be subcontracted.

Acknowledgment

The authors would like to thank Jean-François Côté for repeating the tests of his tabu search and providing the results.

			LNS		**LNS–noSC**		**LNS–noDec**		$\overline{\textbf{LNS}}$	
Inst.	$\lvert\mathcal{N}\rvert$	BKS	$\Delta_{best}(\%)$	t(sec)	$\Delta_{best}(\%)$	t(sec)	$\Delta_{best}(\%)$	t(sec)	f	$\Delta_f(\%)$
CE-H										
CE-H-01	50	**1191.70**	0.19	27	0.19	27	0.19	16	**1191.70**	**0.00**
CE-H-02	75	1790.67	0.46	49	0.46	49	**-0.07**	121	**1789.41**	**-0.07**
CE-H-03	100	1917.96	-0.10	70	-0.10	70	**-0.23**	78	**1913.49**	**-0.23**
CE-H-04	150	2475.16	-0.10	131	0.04	129	-0.01	149	**2465.51**	**-0.39**
CE-H-05	199	3143.01	**-0.76**	322	**-0.76**	274	-0.50	416	**3119.10**	**-0.76**
CE-H-06	50	**1204.48**	0.00	26	0.00	26	0.00	13	**1204.48**	**0.00**
CE-H-07	75	**2025.98**	0.34	53	0.34	53	0.04	103	2026.70	0.04
CE-H-08	100	1984.36	-0.28	71	-0.28	71	0.00	64	**1978.79**	**-0.28**
CE-H-09	150	2438.73	-0.32	130	-0.28	128	-0.42	150	**2424.43**	**-0.59**
CE-H-10	199	3267.85	-0.71	287	-0.58	253	-0.54	387	**3240.00**	**-0.85**
CE-H-11	120	2303.13	**-0.06**	105	**-0.06**	105	0.00	62	**2301.78**	**-0.06**
CE-H-12	100	1908.74	**-0.04**	74	**-0.04**	72	**-0.04**	109	**1908.05**	**-0.04**
CE-H-13	120	2842.18	-0.07	104	-0.07	104	-0.02	59	**2832.88**	**-0.33**
CE-H-14	100	**1907.74**	0.30	71	0.30	71	0.17	68	**1907.75**	**0.00**
G-H										
G-H-01	240	14251.75	-0.81	325	-0.69	303	-0.74	277	**14097.33**	**-0.95**
G-H-02	320	18560.07	-0.57	539	-0.51	400	-0.04	453	**18412.40**	**-0.80**
G-H-03	400	25356.63	**-1.34**	589	-1.19	465	-0.45	503	**25016.72**	**-1.34**
G-H-04	480	34589.11	**-0.75**	774	-0.65	553	0.85	627	**34328.99**	**-0.75**
G-H-05	200	15667.13	-1.10	184	-1.04	182	-0.25	99	**15398.76**	**-1.71**
G-H-06	280	19975.32	-0.74	337	-0.70	301	-0.60	306	**19743.63**	**-1.16**
G-H-07	360	23510.98	-0.04	531	0.08	398	0.73	431	**23293.54**	**-0.92**
G-H-08	440	27420.68	**-0.23**	659	-0.13	496	0.65	554	**27358.69**	**-0.23**
G-H-09	255	1331.83	0.10	468	0.10	331	0.24	448	**1324.99**	**-0.27**
G-H-10	323	1561.52	0.23	596	0.23	425	-0.02	463	**1556.39**	**-0.05**
G-H-11	399	2195.31	**-0.30**	935	-0.21	638	0.37	524	**2185.08**	**-0.30**
G-H-12	483	**2487.38**	0.36	1176	0.36	873	1.03	559	2488.08	0.03
G-H-13	252	2239.18	-0.42	727	-0.29	568	0.05	519	**2218.92**	**-0.85**
G-H-14	320	2682.85	-0.88	1156	-0.88	916	-0.17	625	**2649.32**	**-1.25**
G-H-15	396	3131.89	-0.39	1705	-0.35	1403	0.55	681	**3108.53**	**-0.45**
G-H-16	480	3629.41	**-0.60**	2111	**-0.60**	1807	0.77	738	**3598.41**	**-0.60**
G-H-17	240	1695.75	**-0.58**	571	-0.38	376	-0.29	460	**1685.97**	**-0.58**
G-H-18	300	2740.05	-0.37	1004	-0.27	701	0.46	590	**2729.61**	**-0.38**
G-H-19	360	3464.70	-0.22	1451	-0.16	1145	0.23	675	**3453.41**	**-0.33**
G-H-20	420	4352.35	-0.80	2129	-0.80	1820	0.45	798	**4311.17**	**-0.95**
Avg.			**-0.31**	**573**	**-0.26**	**457**	**0.07**	**357**		**-0.51**

Table 3: Detailed results and comparison of algorithmic components on the instance sets with a heterogeneous fleet.

	RIP	**TS**	**TS+**	**AVNS**	**AVNS-RN**	**UHGS**	**MS-ILS**	**MS-LS**	**LNS**
Avg. $\Delta_{best}(\%)$									
CE	1.063	0.345	0.309	0.196	0.131	0.015	0.065	1.405	0.008
G	1.988	1.807	†0.225	0.636	0.554	0.110	0.469	2.643	0.181
CE & G	1.666	1.258	†0.266	0.473	0.380	0.074	0.317	2.211	0.110
Avg. t(min)	1×17.45	10×2.98	10×34.86	10×11.94	10×12.14	10×26.40	10×16.62	10×1.89	10×9.49
Avg. $\Delta_{best}(\%)$									
CE-H	0.646	0.391	0.291						-0.082
G-H	1.324	1.162	†-0.344						-0.473
CE-H & G-H	1.084	0.879	†-0.093						-0.305
Avg. t(min)	1×17.50	10×2.86	10×40.56						10×9.55
CPU@GHz	Xeon@3.6	I7@3.4	Opteron@2.2	I5@2.67	I5@2.67	Xeon@3.07	Xeon@3.07	Xeon@3.07	I7@2.8

Table 4: Overview of results obtained with LNS and heuristics from the literature.

Inst.	$\|\mathcal{N}\|$	IUB	LB	$\Delta_{LB}(\%)$	t(sec)	#Nds	#Cuts
CE							
CE-01	50	1119.47	**1119.47**	0.00	10.7	27	127
CE-02	75	1814.52	**1814.52**	0.00	489.7	224	227
CE-03	100	1919.05	1916.49	0.13	TL	237	240
CE-04	150	2505.39	2495.26	0.41	TL	312	289
CE-05	199	3081.59	3066.67	0.49	TL	265	339
CE-06	50	1207.47	**1207.47**	0.00	13.9	25	147
CE-07	75	2004.53	**2004.53**	0.00	280.8	154	229
CE-08	100	2052.05	2049.11	0.14	TL	219	233
CE-09	150	2419.84	2407.18	0.53	TL	257	267
CE-10	199	3376.80	3355.75	0.63	TL	240	270
CE-11	120	2330.94	2323.89	0.30	TL	5	21
CE-12	100	1952.86	1950.79	0.11	TL	217	225
CE-13	120	2858.83	2850.75	0.28	TL	7	21
CE-14	100	2213.02	2211.63	0.06	TL	149	225
G							
G-01	240	14131.20	14044.00	0.62	TL	150	225
G-02	320	19142.80	18987.20	0.82	TL	56	225
G-03	400	24409.00	24200.60	0.86	TL	1	0
G-04	480	34183.10	33898.40	0.84	TL	1	0
G-05	200	14223.60	14144.50	0.56	TL	33	225
G-06	280	21382.20	21299.60	0.39	TL	29	225
G-07	360	23373.40	23172.40	0.87	TL	2	0
G-08	440	29679.80	29439.60	0.82	TL	1	0
G-09	255	1325.03	1308.85	1.24	TL	11	60
G-10	323	1586.50	1567.65	1.20	TL	1	0
G-11	399	2163.72	2137.78	1.21	TL	1	0
G-12	483	2490.23	—	—	TL	0	0
G-13	252	2258.02	2234.83	1.04	TL	44	225
G-14	320	2682.90	2658.79	0.91	TL	49	225
G-15	396	3145.11	3109.45	1.15	TL	16	60
G-16	480	3614.79	3578.88	1.00	TL	2	0
G-17	240	1666.31	**1666.31**	0.00	43.5	1	0
G-18	300	2730.55	2719.11	0.42	TL	270	225
G-19	360	3492.31	3479.43	0.37	TL	154	225
G-20	420	4303.56	4278.97	0.57	TL	47	225

Table 5: Results of the BPC for the homogeneous instances.

Inst.	$\lvert\mathcal{N}\rvert$	IUB	LB	$\Delta_{LB}(\%)$	t(sec)	#Nds	#Cuts
CE-H							
CE-H-01	50	1191.70	**1191.70**	0.00	598.4	78	226
CE-H-02	75	1790.67	***1789.41**	0.00	2773.3	604	226
CE-H-03	100	1913.49	1906.78	0.35	TL	91	233
CE-H-04	150	2465.51	2449.38	0.66	TL	140	225
CE-H-05	199	3119.10	3095.40	0.77	TL	129	225
CE-H-06	50	1204.48	**1204.48**	0.00	56.7	36	186
CE-H-07	75	2025.98	***2025.05**	0.00	5426.8	762	225
CE-H-08	100	1978.79	1973.81	0.25	TL	163	225
CE-H-09	150	2424.43	2414.22	0.42	TL	198	225
CE-H-10	199	3240.00	3218.85	0.66	TL	104	225
CE-H-11	120	2301.78	2286.11	0.69	TL	3	10
CE-H-12	100	1908.05	1903.41	0.24	TL	118	225
CE-H-13	120	2832.88	2824.19	0.31	TL	3	10
CE-H-14	100	1907.74	**1907.74**	0.00	2088.6	78	225
G-H							
G-H-01	240	14097.30	14002.80	0.67	TL	83	225
G-H-02	320	18412.40	18218.10	1.07	TL	15	101
G-H-03	400	25016.70	24729.70	1.16	TL	1	0
G-H-04	480	34329.00	33407.20	2.76	TL	1	0
G-H-05	200	15398.80	15370.10	0.19	TL	69	225
G-H-06	280	19743.60	19595.60	0.76	TL	8	70
G-H-07	360	23293.50	23158.40	0.58	TL	1	0
G-H-08	440	27358.70	26864.30	1.84	TL	1	0
G-H-09	255	1324.99	1308.17	1.29	TL	1	0
G-H-10	323	1556.39	1530.34	1.70	TL	1	0
G-H-11	399	2185.08	2152.47	1.52	TL	1	0
G-H-12	483	2488.08	—	—	TL	0	0
G-H-13	252	2218.92	2194.40	1.12	TL	47	225
G-H-14	320	2649.32	2621.75	1.05	TL	16	50
G-H-15	396	3108.53	3059.82	1.59	TL	9	40
G-H-16	480	3598.41	3547.73	1.43	TL	1	0
G-H-17	240	1685.97	1682.40	0.21	TL	320	225
G-H-18	300	2729.61	2718.39	0.41	TL	165	225
G-H-19	360	3453.41	3430.93	0.66	TL	67	225
G-H-20	420	4311.17	4270.90	0.94	TL	28	151

Table 6: Results of the BPC for the heterogeneous instances.

References

R. Baldacci, A. Mingozzi, and R. Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.

M.-C. Bolduc, J. Renaud, and F. Boctor. A heuristic for the routing and carrier selection problem. *European Journal of Operational Research*, 183:926–932, 2007.

M.-C. Bolduc, J. Renaud, F. Boctor, and G. Laporte. A perturbation metaheuristic for the vehicle routing problem with private fleet and common carriers. *Journal of the Operational Research Society*, 59:776–787, 2008.

N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.

C.-W. Chu. A heuristic algorithm for the truckload and less-than-truckload problem. *European Journal of Operational Research*, 165:657–667, 2005.

G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

C. Contardo and R. Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129–146, 2014.

J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.

J.-F. Côté and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with private fleet and common carrier. *European Journal of Operational Research*, 198:464–469, 2009.

D. Feillet, M. Gendreau, and L.-M. Rousseau. New refinements for the solution of vehicle routing problems with branch and price. *INFOR*, 45(4):239–256, 2007.

T. J. Gaskell. Bases for vehicle fleet scheduling. *Journal of the Operational Research Society*, 18(3):281–295, 1967.

M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

B. Golden, E. Wasil, J. Kelly, and I.-M. Chao. Fleet management and logisitics. In T. G. Grainic and G. Laporte, editors, *The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results*, pages 33–56. Springer, 1998.

C. Groër, B. Golden, and E. Wasil. A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing*, 23(2):315–330, 2011.

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

S. Irnich, G. Desaulniers, J. Desrosiers, and A. Hadjar. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, 22(2):297–313, 2010.

M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

M. A. Krajewska and H. Kopfer. Transportation planning in freight forwarding companies: Tabu search algorithm for the integrated operational transportation planning problem. *European Journal of Operational Research*, 197(2): 741–751, 2009.

F. Li, B. Golden, and E. Wasil. Very large-scale vehicle routing: New test problems, algorithms, and results. *European Journal of Operational Research*, 32:1165–1179, 2005.

R. Liu, Z. Jiang, X. Liu, and F. Chen. Task selection and routing problems in collaborative truckload transportation. *Transportation Research Part E: Logistics and Transportation Review*, 46(6):1071–1085, 2010.

M. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

J. Lysgaard. Cvrpsep: A package of separation routines for the capacitated vehicle routing problem. Working Paper 03-04, Department of Management Science and Logistics, Aarhus School of Business, Aarhus, Denmark, 2003.

D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

J.-Y. Potvin and M.-A. Naud. Tabu search with ejection chains for the vehicle routing problem with private fleet and common carrier. *Journal of the Operational Research Society*, 62:326–336, 2011.

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

R. Roberti and A. Mingozzi. Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, 48 (3):413–424, 2014.

Y. Rochat and É. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.

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

S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775, 2006b.

M. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, Department of Computer Science, University of Strathclyde, 1997.

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

A. Stenger, M. Schneider, and D. Goeke. The prize-collecting vehicle routing problem with single and multiple depots and non-linear cost. *EURO Journal on Transportation and Logistics*, 2(1-2):57–87, 2013a.

A. Stenger, D. Vigo, S. Enz, and M. Schwind. An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping. *Transportation Science*, 47(1):64–80, 2013b.

A. Subramanian, E. Uchoa, and L. S. Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531, 2013.

L. Tang and X. Wang. Iterated local search algorithm based on very large-scale neighborhood for prize-collecting vehicle routing problem. *The International Journal of Advanced Manufacturing Technology*, 29(11):1246–1258, 2006.

C. Tilk, A.-K. Rothenbächer, T. Gschwind, and S. Irnich. Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, 261(2):530–539, 2017.

T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1–21, 2013.

T. Vidal, N. Maculan, L. S. Ochi, and P. H. V. Penna. Large neighborhoods with implicit customer selection for vehicle routing problems with profits. *Transportation Science*, 50(2):720–734, 2016.

C. D. J. Waters. A solution procedure for the vehicle-scheduling problem based on iterative route improvement. *Journal of the Operational Research Society*, 38(9):833–839, 1987.

P. Yellow. A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly*, 21:281–283, 1970.

A Appendix

Table 7 provides updated results for the TS introduced in Côté and Potvin (2009). The coordinates of customers in instances of sets G and G-H are specified with a precision of four decimal places, but the results reported in Côté and Potvin (2009) are obtained treating them as integers. The authors repeated their tests with the higher precision and provided us with the results in order to make the TS comparable with other methods. Columns denoted f_{best} report the best objective value of TS obtained in 10 runs and columns t(sec) the total run-time in seconds to perform 10 runs using an I7 processor with 3.3 GHz. Note that the experiments in Potvin and Naud (2011) were also performed with integer coordinates as they are based on the same code. However, we cannot provide updated results for TS+ because we could not reach the author with the latest version of the code.

	TS			TS	
Inst.	f_{best}	t(sec)	Inst.	f_{best}	t(sec)
CE			**CE-H**		
CE-01	1119.47	75	CE-H-01	1191.70	80
CE-02	1816.28	96	CE-H-02	1792.36	97
CE-03	1922.19	207	CE-H-03	1918.47	208
CE-04	2529.40	350	CE-H-04	2481.46	356
CE-05	3113.33	487	CE-H-05	3150.06	464
CE-06	1207.47	77	CE-H-06	1208.08	77
CE-07	2006.52	95	CE-H-07	2029.40	93
CE-08	2065.95	208	CE-H-08	1989.61	219
CE-09	2437.76	331	CE-H-09	2451.16	332
CE-10	3406.67	491	CE-H-10	3270.25	496
CE-11	2332.03	307	CE-H-11	2335.25	302
CE-12	1952.86	154	CE-H-12	1912.47	154
CE-13	2862.16	307	CE-H-13	2871.79	304
CE-14	2219.31	167	CE-H-14	1925.46	167
G			**G-H**		
G-01	14284.07	1306	G-H-01	14233.25	1345
G-02	19675.29	2484	G-H-02	18727.91	2618
G-03	25543.93	5918	G-H-03	26042.63	4895
G-04	36221.59	8481	G-H-04	36518.80	7677
G-05	14866.58	1917	G-H-05	15897.83	1530
G-06	22455.49	3384	G-H-06	20582.61	2585
G-07	24203.51	4794	G-H-07	24228.65	4436
G-08	30822.56	6154	G-H-08	28505.81	6413
G-09	1326.26	1085	G-H-09	1328.61	1079
G-10	1593.79	1733	G-H-10	1557.13	2036
G-11	2173.82	3073	G-H-11	2191.75	3018
G-12	2494.56	4901	G-H-12	2501.02	4836
G-13	2274.56	596	G-H-13	2237.86	657
G-14	2702.49	1066	G-H-14	2683.88	1124
G-15	3162.90	1921	G-H-15	3122.68	1989
G-16	3643.39	3002	G-H-16	3620.27	3123
G-17	1674.64	563	G-H-17	1702.66	607
G-18	2752.80	919	G-H-18	2754.05	940
G-19	3523.57	1654	G-H-19	3493.83	1657
G-20	4355.19	2378	G-H-20	4360.03	2383

Table 7: Updated results for the TS of Côté and Potvin (2009).