TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

An adaptive large neighborhood search with path relinking for a class of vehicle-routing problems with simultaneous pickup and delivery

Julian Hof and Michael Schneider

Technical Report BISOR-01/2017

Julian Hof
Chair of Business Information Systems and Operations Research
University of Kaiserslautern, Germany
julian.hof@wiwi.uni-kl.de

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

Abstract

We study a class of vehicle-routing problems (VRP) with simultaneous pickup and delivery (VRPSPD). In VRPSPDs, each customer may require a certain quantity of goods delivered from the depot and a quantity of goods to be picked up and returned to the depot. Besides the standard VRPSPD, we address (i) the VRPSPD with time limit (VRPSPDTL), which imposes a time limit on the routes of the transportation vehicles, (ii) the VRPSPD with time windows (VRPSPDTW), which takes customer time windows into account, (iii) the VRP with divisible deliveries and pickups (VRPDDP), which allows for fulfilling the delivery and pickup requests of each customer in two separate visits, and (iv) the previously unstudied VRPDDP with time windows (VRPDDPTW). We develop a hybrid heuristic solution method which combines an adaptive large neighborhood search algorithm with a path relinking approach, called ALNS-PR, and we demonstrate the competitiveness of our algorithm on benchmark instances proposed in the literature.

Keywords: vehicle routing, simultaneous pickup and delivery, large neighborhood search, path relinking

1 Introduction

In this paper, we study a class of vehicle-routing problems (VRP) with simultaneous pickup and delivery (VRPSPD) where customers may require (i) delivery service of goods originating at the depot, and (ii) pickup service for goods which need to be returned to the depot. VRPSPDs arise, for instance, in the context of reverse logistics, defined as the process of planning, implementing, and controlling the flow of, e.g., raw materials or finished goods from the point of consumption to the point of origin for the purpose of recapturing value or proper disposal by The Council of Logistics Management (Stock 1992). In particular, this paper is motivated by a practical application occurring at DHL Freight in Sweden, in which customers requiring bulky goods to be delivered and to be picked up need to be served on a daily basis. In a first step, we aim at developing a solution method that is flexible enough to address the VRPSPD and its variants from the literature, and that provides competitive results on these problems.

In addition to the standard VRPSPD, we investigate (i) the VRPSPD with time limit (VRPSPDTL), which imposes a maximum duration on the vehicle routes, (ii) the VRPSPD with time windows (VRPSPDTW), where customers are associated with time intervals in which their service must start, (iii) the VRP with divisible deliveries and pickups (VRPDDP), which allows to satisfy a customer's pickup and delivery requests in two separate visits, and (iv) the previously unstudied VRPDDP with time windows (VRPDDPTW).

Because the standard VRPSPD and its variants extend the $\mathcal{NP}$-hard Capacitated VRP (CVRP), heuristic solution methods are a suitable choice for solving larger problem instances of practical relevance in short computation time.

Several heuristics have been proposed for the VRPSPD. The most successful ones are the parallel iterated local search algorithm by Subramanian et al. (2010), that makes use of the variable neighborhood descent paradigm with random neighborhood ordering and is embedded in a multi-start framework, the approach of Zachariadis et al. (2010), which is characterized by a memory structure to store and recombine promising vertex sequences, the iterated local search of Subramanian et al. (2013) that is combined with a set partitioning approach, the unified hybrid genetic search by Vidal et al. (2014), and the variable neighborhood search (VNS) algorithm extended by a perturbation mechanism proposed by Polat et al. (2015). The approaches of Subramanian et al. (2013) and Polat et al. (2015) are also the best-performing heuristics for the VRPSPDTL.

Wang and Chen (2012) develop a co-evolution genetic algorithm which is evaluated on VRPSPDTW instances that are derived from instances for the VRP with time windows (VRPTW). Wang et al. (2015) are able to improve on numerous results reported by Wang and Chen (2012) using a parallel simulated annealing (SA) algorithm.

Nagy et al. (2015) introduce the VRPDDP and use a reactive tabu search algorithm to investigate the conditions under which the division of customer demands is beneficial. Recently, Polat (2017) presented cooperative VNS, a parallel approach based on the VNS paradigm for the VRPDDP, which is able to obtain new best solutions for the large majority of the instances proposed by Nagy et al. (2015).

Ropke and Pisinger (2006b) proposed an adaptive large neighborhood search (ALNS) algorithm for a class of VRPs with backhauls that was also applied to a small subset of VRPSPD instances but could not compete with the state-of-the-art approaches specifically tailored to the VRPSPD. The contribution of this paper is to propose a more effective ALNS heuristic combined with a path relinking (PR) approach, called ALNS-PR, to address the VRPSPD and its variants. We introduce an innovative ALNS operator that takes the load characteristics of VRPSPDs into account and significantly accelerates the convergence rate of the search.

To evaluate the performance of our ALNS-PR algorithm, we perform extensive computational studies and demonstrate the competitiveness of our approach on established benchmark instances proposed in the literature. We improve numerous previous best-known solutions (BKS), especially on VRPSPDTL, VRPSPDTW, and VRPDDP instances. In addition, we introduce a new VRPDDPTW benchmark and report results on those instances.

The paper is structured as follows: Our ALNS-PR algorithm is explained in detail in Section 2. We describe the numerical studies to evaluate the performance of the proposed approach in Section 3. The paper is summarized and concluded in Section 4.

2 Adaptive large neighborhood search with path relinking

In this section, we describe our solution method that combines an ALNS algorithm with a PR approach. A pseudocode overview of the ALNS-PR is given in Figure 1.

```
$S \leftarrow generateInitialSolution()$
Initialize best solution $S^* \leftarrow S$
**while** number of iterations without improvement not reached **do**
  **if** $S$ already visited or set of elite solutions $\mathcal{E}$ not completely filled **then**
    {*Perform single ALNS iteration and local search on subset $\mathcal{K}$ of routes drawn from $S$*}
    $S' \leftarrow performALNS(S, \mathcal{K})$
    $S' \leftarrow performLocalSearch(S', \mathcal{K})$
  **else**
    $S' \leftarrow performPathRelinking(S, \mathcal{E})$
    $S' \leftarrow performLocalSearch(S')$
  **end if**
  **if** accept$(S', S)$ **then**
    $S \leftarrow S'$
    **if** $S'$ improves on $S^*$ **then**
      $S^* \leftarrow S'$
    **end if**
  **end if**
  $updatePenaltyFactors(S)$
**end while**
```

Figure 1: Pseudocode of the ALNS-PR algorithm

The initial solution S is generated by means of the savings algorithm introduced by Clarke and Wright (1964) (Section 2.2) and improved by a local search procedure (Section 2.5).

The following main phase of our algorithm is then repeated until a maximum number of iterations ω without improvement of the current best solution S^* is reached. In each iteration, we decide whether to apply the ALNS or the PR component to the current solution S. Our PR component relies on a dynamic set $\mathcal{E}$ of elite solutions that are recombined with new solutions found during the search. If the current solution S has already been visited before, or the elite set is not completely filled, i.e., $|\mathcal{E}|$ is smaller than the maximum size λ of the elite set, a single iteration of our ALNS component is performed. We apply ALNS and the following local search step only to a subset $\mathcal{K}$ of routes drawn from the current solution S that are closely located to each other (Section 2.3), resulting in solution S'. We observed a beneficial impact on solution quality and run-time compared to the application of both components to the entire solution.

Otherwise, if S represents a new solution and the elite set $\mathcal{E}$ is filled, PR is applied between S and the solutions contained in $\mathcal{E}$ (Section 2.4). Subsequently, we perform local search on the best solution S' returned by the PR routine.

For the resulting solution S', we evaluate if it should be included in the set of elite solutions $\mathcal{E}$. As long as $\mathcal{E}$ is not completely filled, any feasible solution is added. Otherwise, S' is included and replaces the worst solution contained in $\mathcal{E}$ if it is feasible, and

- improves on the best solution in $\mathcal{E}$, or
- a SA-based comparison with the worst solution in $\mathcal{E}$ similar to the mechanism described in Section 2.6 turns out in favor of the candidate solution S', and the inclusion of S' does not decrease the average diversity among all elite solutions. We measure the diversity between solutions in terms of the number of arcs exclusively contained in one of both solutions (see Section 2.4).

This procedure aims at balancing quality and diversity of the elite solutions.

Next, the SA acceptance mechanism decides if S' replaces S as the current solution for the subsequent iteration (Section 2.6). After μ iterations without improvement, we reset S to S^*. For the purpose of diversification, we set the SA temperature back to its initial value and S to a solution randomly chosen from the elite set after ϵ solution resets. The probability of an elite solution to be selected is proportional to the diversity between this solution and S^*.

Infeasible solutions, i.e., solutions not respecting all constraints are allowed and handled by means of a dynamic penalty mechanism. More precisely, we transform constraint violations into penalty costs by multiplying the respective value with a dedicated penalty factor for each constraint. The penalty factors of solution S are dynamically updated during the search depending on the number of consecutive ALNS-PR iterations during which the respective constraint has been satisfied or violated (Section 2.1). Moreover, if the ALNS component has been applied in the current iteration, we update the selection probabilities of its components according to the adaptive mechanism described in Section 2.3.

Heuristic optimization for the classical VRPTW is usually conducted under the assumption of a hierarchical objective function with the main goal of minimizing the number of employed vehicles (Bräysy and Gendreau 2005). This is also true for the investigated problem variants VRPSPDTW and VRPDDPTW. If customer time windows are present, we enable a vehicle minimization phase if the current solution S is feasible, and the number of unsuccessful vehicle reduction attempts is smaller than a nonnegative integer θ. Then, our ALNS tries to remove a route from the current solution, and in our PR component, we ignore elite solutions that utilize a higher number of vehicles than S for relinking. Otherwise, if S has been infeasible for ι iterations, we add an empty route to S. In the following description of our solution method, we explicitly point out the modifications that become necessary when time windows need to be taken into account.

Finally, to consider the possibility of satisfying a customer's pickup and delivery requests via two separate visits as in the VRPDDP(TW), we extend the previously described algorithm by an additional phase. Any VRPDDP can be transformed into a VRP with mixed deliveries and pickups (VRPMDP) with twice as many customers where each (partial) customer either requires pickup or delivery service but not both. To this end, each customer of an original VRPDDP(TW) instance is replaced by (i) a duplicate of that customer having a delivery demand equal to the delivery demand of the original customer and a pickup demand equal to zero, and (ii) a duplicate having a delivery demand equal to zero and a pickup demand equal to the pickup demand of the original customer.

Naturally, doubling the instance size significantly increases the computational effort required by any solution method. Therefore, we refrain from duplicating each customer at the beginning of the search. Instead, we first apply the main phase of our algorithm as previously described under the assumption of indivisible customer demands. After ω iterations without improvement of the current best VRPSPD(TW) solution S^*, we duplicate each customer in S^* and in each current elite solution. We subsequently repeat the main phase of our algorithm using half of the initial value of ω as stopping condition with the goal of obtaining an improved VRPDDP(TW) solution.

2.1 Solution evaluation and penalty mechanism

In order to increase the flexibility in exploring the solution space, our ALNS-PR temporarily tolerates constraint violations by imposing dynamic penalty costs on infeasible solutions. The objective function value of a solution S is then determined using a generalized cost function $f_{gen}(S)$ which includes penalty costs for violating the capacity and time window constraints:

$$f_{gen}(S) = f_{dist}(S) + \delta^C \upsilon^C(S) + \delta^{TW} \upsilon^{TW}(S).$$

Here, $f_{dist}(S)$ denotes the total traveled distance of solution S. The penalty factor for capacity (time window) violation is denoted as δ^C (δ^{TW}), and the current capacity (time window) violation in solution S is given by $\upsilon^C(S)$ ($\upsilon^{TW}(S)$).

All penalty factors are initially set to δ^0 and dynamically varied within the interval $[\delta^{min}, \delta^{max}]$. After η^+ consecutive ALNS-PR iterations during which S has been infeasible with respect to a certain constraint, the associated penalty factor is increased by factor δ^{update}. Analogously, after η^- iterations without violating a certain constraint, the respective penalty factor is divided by the factor δ^{update}.

In contrast to the classical CVRP, the vehicle load in the VRPSPD and its extensions does not monotonically decrease or increase along a route but rather fluctuates. Therefore, for a solution to the VRPSPD, feasibility with respect to the vehicle capacity is given if for each vertex i of each route, the total demand picked up until vertex i plus the total demand still to be delivered at vertex i does not exceed the vehicle capacity. Thus, even for intra-route moves, capacity evaluations cannot be performed in constant time without introducing additional data structures. We efficiently evaluate capacity violations by implementing similar data structures as described in Zachariadis et al. (2010), which rely on storing forward and backward demand and load quantities for each vertex in a route. Given a route r and let $\mathcal{V}_r$ denote the set of vertices visited by r, Q the vehicle capacity and l_i the vehicle load at vertex i, we define the capacity violation of route r as the maximum violation of the vehicle capacity encountered along all vertices of r:

$$\upsilon^C(r) = \max_{i \in \mathcal{V}_r}(\max(l_i - Q, 0)).$$

Time window violations are calculated based on the principle of time travel as described in Nagata et al. (2010) and Schneider et al. (2013). After considering a time window violation only once at the vertex of occurrence, it is assumed that the vehicle is allowed to perform service at the latest feasible moment, i.e., at the end of the time window of the regarded customer. This procedure ensures that violations are not accumulated along actually feasible succeeding vertex sequences. Using this approach, the computation of changes in time window violation for conventional inter-route moves can be conducted in $\mathcal{O}(1)$.

As stated in the previous section, in the case of VRPSPD variants with customer time windows, we are faced with a hierarchical objective function with the main goal of minimizing the number of employed vehicles. Then, f_{gen} becomes the secondary objective function that needs to be evaluated if two solutions utilize the same number of vehicles.

2.2 Initialization with savings algorithm

In order to quickly generate an initial solution, we implement an adaption of the savings algorithm introduced by Clarke and Wright (1964) consisting of the following steps:

1. Generate back-and-forth tours for all customers.
2. Calculate the cost saving for each pair of customers resulting from merging the associated routes as $s = c_{0i} + c_{0j} - c_{ij}$, where c_{ij} corresponds to the distance between two vertices i and j, and sort the savings in decreasing order.
3. While there are positive cost savings, merge the associated routes if this does not result in any constraint violation. Note that, if both routes either start or end with the customers corresponding to the current saving value, we need to reverse the order of customer visits in one route before the merging can be performed. However, this is usually undesirable in settings where tight time windows need to be considered. In addition, due to the resulting shift of pickup and delivery demands, a reversal of the route direction might render an initially feasible route infeasible with respect to the capacity constraint. Thus, if the route reversal would lead to any constraint violation, the merging is not performed.

After the merging procedure, the resulting number of routes may exceed the number of available vehicles. In this case, we identify the route in which the maximum load encountered along all customer visits is minimum. This route is then dissolved, and each customer inserted at the cheapest position in the remaining routes. Capacity and time window violations are handled by imposing penalty costs according to the generalized cost function (see Section 2.1). We repeat the process of dissolving routes until the number of routes complies with the number of available vehicles.

The obtained solution is subsequently improved by a local search step (see Section 2.5).

2.3 The adaptive large neighborhood search component

The LNS paradigm, originally introduced by Shaw (1998), consists in iteratively destroying and subsequently repairing solutions by removing and reinserting relatively large numbers of customers. ALNS extends this approach by deploying several competing removal and insertion operators which are chosen at each iteration depending on their previous search performance. To this end, each operator is assigned a selection probability which is dynamically updated during the search (Ropke and Pisinger 2006a). ALNS has been successively applied to several VRP variants (see, e.g., Ropke and Pisinger 2006a,b, Pisinger and Ropke 2007, Hemmelmayr et al. 2012).

In our solution approach, the ALNS component mainly serves the purpose of diversifying the search. Figure 2 shows a single iteration of our ALNS implementation in pseudocode.

First, we determine the subset of routes $\mathcal{K}$ from which customers are removed and subsequently reinserted. To this end, a seed route is randomly selected. The remaining routes to be contained in $\mathcal{K}$ are identified by repeating the following procedure: For each route not yet in $\mathcal{K}$, we calculate the average distance of its customers to all customers already contained in $\mathcal{K}$. If this value is smaller than $\varsigma^d \max\limits_{i,j\in\mathcal{V}}(c_{ij})$, where $\varsigma^d \in (0, 1]$ is called distance threshold factor and $\max\limits_{i,j\in\mathcal{V}}(c_{ij})$ corresponds to the

```
$\mathcal{K} \leftarrow getNearestRoutes(S, \boldsymbol{\pi}^d)$
if vehicle minimization enabled then
    $S' \leftarrow removeRoute(S, \mathcal{K})$
else
    $n^- \leftarrow$ drawNumberOfCustomersToRemove($\boldsymbol{\pi}^{||}$, $\mathcal{K}$)
    {Apply randomly selected removal operator based on probabilities $\boldsymbol{\pi}^-$}
    $S' \leftarrow removeCustomers(S, \mathcal{K}, n^-, \boldsymbol{\pi}^-)$
end if
{Apply randomly selected insertion operator based on probabilities $\boldsymbol{\pi}^+$}
$S' \leftarrow insertCustomers(S', \mathcal{K}, \boldsymbol{\pi}^+)$
```

Figure 2: Pseudocode of our ALNS component

maximum distance between any two vertices in the problem instance, the respective route is added to the subset. To increase the flexibility of the subset generation, at each ALNS iteration, ς^d is randomly selected from list $\Psi^d = (\varsigma^d_{min}, \varsigma^d_{min}+0.05, ..., \varsigma^d_{max}-0.05, \varsigma^d_{max})$ with the minimum and maximum values, ς^d_{min} and ς^d_{max}, respectively, being multiples of 0.05. Each value in Ψ^d is assigned a probability from vector $\boldsymbol{\pi}^d$ which is dynamically updated during the search according to the mechanism described in Section 2.3.3.

If the vehicle minimization phase is currently enabled, we use the route removal operator (Section 2.3.1) to remove a route contained in $\mathcal{K}$ from the current solution. Otherwise, we first select the number of customers that are removed from $\mathcal{K}$. In most LNS implementations, this is done by randomly selecting the percentage of customers to remove from a relatively large interval $\Psi^{||} = [\varsigma^{||}_{min}, \varsigma^{||}_{max}]$. However, to account for the observation that ideal removal percentages are highly instance-dependent, we split $\Psi^{||}$ into five sub-intervals as done by Goeke and Schneider (2015). At each ALNS iteration, we select a sub-interval based on dynamic probabilities $\boldsymbol{\pi}^{||}$. The number n^- of customers to remove from $\mathcal{K}$ is then randomly determined within the selected interval. The customer removal is subsequently performed by means of a removal operator that is selected based on probabilities $\boldsymbol{\pi}^-$.

Finally, we select an insertion operator (Section 2.3.1) according to probabilities $\boldsymbol{\pi}^+$ and successively reinsert the previously removed customers into the routes of $\mathcal{K}$.

2.3.1 Removal and insertion operators

To remove customers from the previously determined subset of routes $\mathcal{K}$, we use the following operators:

Random removal randomly removes customers from $\mathcal{K}$ until n^- customers are removed.

Cluster removal was first introduced by Ropke and Pisinger (2006a) and aims at removing customers that are located close to each other. First, we select a route and the first customer to be removed from this route at random. Subsequently, the following steps are repeated until n^- customers are removed from the current solution: We randomly choose a customer among the already removed customers and identify the route r which yields the smallest average distance of its customers to the selected customer. Next, we apply Kruskal's algorithm (Kruskal 1956) for solving minimum-spanning-tree (MST) problems to the sub-graph composed of the customers of route r. Let n_r denote the number of customers served by route r. We abort the execution of the MST algorithm as soon as the number of generated edges is equal to $n_r - 2$, i.e., two sub-trees remain from which one is randomly chosen for removal. If the size of the selected cluster exceeds the number of remaining customers to be removed, we randomly remove customers from the cluster until n^- customers are removed.

Relatedness removal follows the idea of removing customers that are considered similar and thus likely to be interchangeable (Shaw 1997). The relatedness of two customers i and j is measured in terms of the distance c_{ij} between them, the difference in their delivery as well as pickup demands $|d_i^d - d_j^d|$ and $|d_i^p - d_j^p|$, respectively, and the difference between the earliest start times of their time windows $|e_i - e_j|$. Each partial relatedness measure is weighted with a parameter χ and normalized using the respective extreme values across the set of all customers $\mathcal{C}$ given by the problem instance. The relatedness measure $R_{i,j}$ of two customers i and j is thus calculated as follows:

$$R_{i,j} = \chi^c \frac{c_{ij}}{\max\limits_{i,j \in \mathcal{C}}(c_{ij})} + \chi^d \frac{|d_i^d - d_j^d|}{\max\limits_{i \in \mathcal{C}}(d_i^d) - \min\limits_{i \in \mathcal{C}}(d_i^d)} + \chi^p \frac{|d_i^p - d_j^p|}{\max\limits_{i \in \mathcal{C}}(d_i^p) - \min\limits_{i \in \mathcal{C}}(d_i^p)} + \chi^e \frac{|e_i - e_j|}{\max\limits_{i \in \mathcal{C}}(e_i) - \min\limits_{i \in \mathcal{C}}(e_i)}.$$

Initially, we select a route and the first customer to be removed from this route at random. Then, we first randomly select a customer from the already removed customers. Next, all remaining customers in $\mathcal{K}$ are stored in a list of size L in ascending order of their relatedness value with respect to the selected customer. From this list, we draw the customer at position $\lfloor L\zeta^{\chi^{rel}} \rfloor$, where ζ is a random number $\in [0, 1)$ and χ^{rel} a parameter $\geqslant 1$, which allows to balance the influence of randomness and relatedness on the selection. We repeat this procedure until n^- customers are removed.

Worst removal as originally introduced by Ropke and Pisinger (2006a) aims at removing customers which appear to be unfavorably positioned in the current solution with respect to the additional routing cost caused by serving them. All customers contained in the current subset $\mathcal{K}$ are stored in a list of size L and sorted in descending order of the cost reduction resulting when removing them from the current solution. At each iteration, we choose the customer at position $\lfloor L\zeta^{\chi^{worst}} \rfloor$. Again, ζ is a random number $\in [0, 1)$, and $\chi^{worst} \geqslant 1$ allows to control the randomness of the selection.

Neighbor graph removal, proposed in Ropke and Pisinger (2006b), is based on exploiting information about promising orders of customer visits gathered in the course of the search. More precisely, a complete directed and weighted auxiliary graph, called the neighbor graph, is introduced, whose vertices correspond to the customers contained in the current subset of routes $\mathcal{K}$. Each arc (i, j) is weighted with the objective function value of the best solution found so far in which customer j is visited directly after customer i. Initially, the weight of each arc is set to positive infinity and dynamically updated during the search.

All customers in $\mathcal{K}$ are stored in a list of size L and sorted in descending order of a score which is based on the current visiting orders in the routes of $\mathcal{K}$. More precisely, the score for each customer i is calculated by summing up the the weights of the arcs (i^-, i) and (i, i^+) in the neighbor graph, where i^- and i^+ correspond to the predecessor and successor of i, respectively. At each iteration, the operator removes the customer at position $\lfloor L\zeta^{\chi^{nb}} \rfloor$ with random number $\zeta \in [0, 1)$ and $\chi^{nb} \geqslant 1$. After the removal of customer i, the neighbor scores of customers i^- and i^+ are updated accordingly.

Route removal removes all customers from the route in $\mathcal{K}$ with the smallest maximum load encountered along all customers. As stated before, we exclusively apply this removal operator if the vehicle minimization phase is currently enabled.

Load balance removal aims at specifically taking the load characteristics of the VRPSPD and its extensions into account. Intuitively, customers with high pickup demands should be visited late while customers with high delivery demands should be served early in a route. We try to identify and rearrange customer visits which seem to be unfavorably positioned with respect to the associated demand quantities and the capacity utilization in the corresponding route. To this end, we first select a route r from $\mathcal{K}$ at random. Next, we calculate a position score $\o_i^{pos}$ for each customer i in r:

$$\o_i^{pos} = p_i \frac{d_i^p}{\sum_{j\in\mathcal{V}_r} d_j^p} + (|\mathcal{V}_r| - 1 - p_i)\frac{d_i^d}{\sum_{j\in\mathcal{V}_r} d_j^d},$$

where p_i corresponds to the current insertion position of customer i. All customers contained in route r are stored in a list of size L in ascending order of their position score. We subsequently draw the customer from position $\lfloor L\zeta^{\chi^{load}}\rfloor$. Analogous to the previous operators, ζ corresponds to a random number $\in [0,1)$, and $\chi^{load} \geqslant 1$ controls the influence of the position score on the customer selection.

To reinsert the previously removed customers into the routes of subset $\mathcal{K}$, we use one of the following insertion operators:

Greedy insertion is implemented in two variants:

1. At each iteration, the best insertion is determined for each of the remaining customers. The customer associated with the overall best insertion is inserted accordingly.
2. The procedure is similar to the first variant except that we do not allow the reinsertion of a customer into the route it had been removed from in the previous removal step. A comparable approach called greedy insertion forbidden is used by Hemmelmayr et al. (2012).

GRASP insertion is inspired by the metaheuristic introduced in Feo and Resende (1989). The best insertion is determined for each remaining customer and stored in a list of size L which is sorted in ascending order of cost increase. At each iteration, the next customer to be inserted is randomly selected among the best $\lfloor \chi^{GRASP} L\rfloor$ remaining insertions. Here, χ^{GRASP} corresponds to a number $\in (0,1)$.

Regret insertion was also proposed by Ropke and Pisinger (2006a) and tries to overcome the myopic behavior of greedy insertion by implementing a look-ahead strategy. More specifically, a regret-k value is calculated for each customer as the difference between the cost increase resulting from the cheapest insertion of this customer into the k-best route and its optimal insertion into the best route from $\mathcal{K}$. At each iteration, we perform the best insertion of the customer associated with the largest regret value. We implement regret insertion for $k = 2, 3, 4$.

Random insertion aims at solution diversification by inserting each customer at a random position of a randomly selected route from $\mathcal{K}$.

2.3.2 Route selection and evaluation of removal and insertion

While some of the previously described removal operators rely on removing customers without considering the properties of the routes they belong to, we additionally use operators which aim at removing customers from routes with certain characteristics. To this end, we use various route selection policies which are combined with the corresponding removal operators (see Schneider et al. 2015, for a similar approach).

Besides a purely random selection, we implement the following policies that determine routes according to a roulette wheel selection mechanism based on specific criteria:

Cost selects a route with a probability proportional to the cost of the route including penalty costs.

Distance is based on the length of a route. The selection probability of a route increases with the associated traveled distance.

Efficiency aims at identifying inefficient routes, i.e., routes that are characterized by large detours to cover a relatively small quantity of customer demands. The probability of a route to be selected is proportional to the ratio of the associated traveled distance and the average capacity utilization in this route.

High average utilization selects a route with a probability proportional to the average capacity utilization in the route.

Low average utilization determines a selection probability for each route that is inversely proportional to the respective average capacity utilization.

High maximum load performs the route selection according to a probability that increases with the maximum load encountered along all customers served in a route.

Low maximum load selects each route based on a probability inversely proportional to the maximum load encountered along all customers of the route.

The last four selection policies focus on the utilization of the vehicle capacity or the fluctuating nature of the vehicle load along routes arising in the investigated problem variants. Obviously, both variants of the average utilization and maximum load policies pursue opposing strategies. However, the adaptive mechanism of our ALNS component is able to successfully identify the appropriate strategies for the problem instance at hand, see Section 3.3.

With respect to compatibility, random removal, cluster removal, and relatedness removal are able to make use of each selection policy. The load balance removal operator exclusively employs the high average utilization policy. Once a compatible removal operator has been selected, we subsequently select a policy based on probabilities $\boldsymbol{\pi}^r$. The selected route selection policy is used by the current removal operator for the entire iteration.

Moreover, to guide the search towards yet unexplored areas, we implement two additional variants of worst removal, greedy insertion, GRASP insertion, and regret insertion that differ in how the corresponding operation is evaluated. Besides the basic operator variants that evaluate the associated operation based on the change in routing cost according to the generalized cost function (Δf_{gen}), we implement variants that employ the evaluation measures:

1. *diversification* Δf_{div}, which additionally considers a diversification penalty based on historic arc occurrence frequencies:

$$\Delta f_{div}(o,S) = \Delta f_{gen}(o,S) + \kappa \frac{f_{gen}(S)}{n} \sqrt{\sum_{(i,j)\in\mathcal{A}_o^+} h(i,j)},$$

   where $\Delta f_{gen}(o,S)$ corresponds to the change of the objective function value of solution S caused by removal or insertion operation o, κ to a real-valued parameter to control the amount of diversification, $f_{gen}(S)$ to the current objective function value of solution S, n to the number of customers given by the problem instance, $h(i,j)$ represents the occurrence frequency of arc (i,j) in previously generated solutions, and $\mathcal{A}_o^+$ the set of arcs generated by operation o, and

2. *noise* Δf_{noise}, which multiplies the change in routing cost caused by operation o by a random number ζ drawn from the interval $[\zeta_{min}, \zeta_{max}]$ (see Ropke and Pisinger 2006a, Hemmelmayr et al. 2012, for a similar approach):

$$\Delta f_{noise}(o,S) = \Delta f_{gen}(o,S)\zeta.$$

2.3.3 Adaptive mechanism

At each ALNS iteration, the choice of distance threshold factor, removal interval, removal operator, route selection policy (if applicable), and insertion operator is performed according to a roulette wheel selection procedure as proposed in Ropke and Pisinger (2006a) based on the probability vectors $\boldsymbol{\pi}^d$, $\boldsymbol{\pi}^{||}$, $\boldsymbol{\pi}^-$, $\boldsymbol{\pi}^r$ and $\boldsymbol{\pi}^+$, respectively. More precisely, given a set of adaptive components denoted as $\mathcal{X} \in \{d, ||, -, r, +\}$, the selection probability of component $i \in \mathcal{X}$ is calculated as $\pi_i = w_i / \sum_{j \in \mathcal{X}} w_j$, where w_i corresponds to the weight of component i. All components of a set $\mathcal{X}$ are initially assigned the same weight and dynamically updated during the search depending on their performance. The performance of an adaptive component is measured in terms of a scoring system. A score of σ^{best} is added to the current score of a component whenever a new overall best solution is found, a score of σ^{imp} if the new solution S' improves on the current one S and has never been encountered before, and a score of σ^{acc} if S' is worse than S but accepted according to the SA acceptance mechanism and has never been encountered before in the search.

We maintain a solution memory to keep track of already obtained solutions and allow our ALNS-PR to fill this memory for ν iterations before any weight update takes place. The weight of each component is then updated every γ ALNS iterations based on its performance during this period. If $\o_i$ denotes the current score of component i and β_i the number of applications of the component since the last weight update, then the new weight is determined as $w_i = w_i(1 - \alpha) + \alpha \o_i / \beta_i$. The factor $\alpha \in [0, 1]$ allows to control the reaction speed of the weight adjustment to the performance of the component. The values of $\o_i$ and β_i are reset to zero after each update.

2.4 The path relinking component

PR represents an intensification strategy which was originally proposed by Glover (1997) and aims at discovering promising solutions on the trajectories between elite solutions obtained during the search. Based on the assumption that good solutions are likely to share common characteristics, PR consists in creating a path of solutions between an initial and a guiding solution in the hope of obtaining improving solutions in the process. The initial solution is thereby transformed into the guiding solution by successively incorporating characteristics of the guiding solution, thus stepwise decreasing the diversity between them.

The PR component complements our ALNS, which primarily focuses on diversifying the search. By connecting promising solutions located in distant regions of the search space, we aim at discovering improving solutions that are not reachable via local search. Ho and Gendreau (2006), Fallahi et al. (2008), and Nguyen et al. (2012), for instance, show how to successfully hybridize well-known metaheuristics for routing problems with PR approaches. Figure 3 shows our PR implementation in pseudocode.

Whenever a solution S has been encountered for the first time in the search, it is passed to the PR algorithm by the main ALNS-PR routine. For each solution S_e contained in the elite set $\mathcal{E}$, we set the current solution on the solution path S_c to the initial solution S and the guiding solution S_g to S_e. In general, we perform PR between S and all elite solutions in $\mathcal{E}$. However, if we are currently trying to reduce the number of vehicles, only solutions in the elite set with a number of routes less than or equal to the number of routes in S are considered for relinking. Otherwise, if S_g utilizes a higher number of vehicles, we need to equalize both vehicle numbers by adding empty routes to S accordingly.

```
$S' \leftarrow \varnothing$
for each solution $S_e \in \mathcal{E}$ do
    {Initialize current and guiding solution}
    $S_c \leftarrow S$
    $S_g \leftarrow S_e$
    {Determine arcs exclusively contained in guiding solution and initialize path length}
    $\mathcal{A}^{\neq} \leftarrow$ getDifferingArcs($S$, $S_g$)
    $p \leftarrow |\mathcal{A}^{\neq}|$
    while $|\mathcal{A}^{\neq}| \geqslant \lfloor (1-\rho) \cdot p \rfloor$ do
        $m_{best} \leftarrow \varnothing$
        for each arc $(i,j) \in \mathcal{A}^{\neq}$ do
            {Determine move which creates arc yielding the least cost change}
            $m \leftarrow$ getBestMoveToCreateArc($i,j$)
            if $m_{best} = \varnothing \vee m$ improves on $m_{best}$ then
                $m_{best} \leftarrow m$
            end if
        end for
        $S_c \leftarrow$ performMove($S_c$, $m_{best}$)
        if $S_c \neq S \wedge S_c \neq S_g \wedge (S' = \varnothing \vee S_c$ improves on $S')$ then
            $S' \leftarrow S_c$
        end if
        {Remove arcs created by previously performed move from set of differing arcs}
        $\mathcal{A}^{\neq} \leftarrow \mathcal{A}^{\neq} \backslash \mathcal{A}^{+}_{m_{best}}$
    end while
end for
```

Figure 3: Pseudocode of our PR component

We explain the following steps of our PR implementation using Figure 4, which illustrates the transformation of an initial solution into a guiding solution for an example instance composed of eleven customers. Both solutions consist of five routes each starting and ending with the depot vertex 0. Initially, we determine those arcs that are contained in S_g but not in the initial solution S and store them in set $\mathcal{A}^{\neq}$. The goal of our PR procedure is now to iteratively incorporate the arcs in $\mathcal{A}^{\neq}$ into the current solution. In our example, there are ten such differing arcs which are exclusively contained in S_g (marked as ✗).

During the execution of PR, we avoid destroying matching arcs, i.e., arcs contained in both solutions. To this end, we connect the vertices incident to matching arcs to fixed vertex sequences that are treated as atomic units and not allowed to be split. Initially, there are two fixed sequences for each customer i, $\Gamma^-_i = (y, ..., i)$ and $\Gamma^+_i = (i, ..., z)$, corresponding to the fixed vertex sequence ending with customer i and the fixed vertex sequence starting with customer i, respectively. Both sequences associated with a customer only contain this customer at the beginning. Consequently, for customer 3 that is served by route A in Figure 4, $\Gamma^-_3 = \Gamma^+_3 = (3)$ initially holds.

Before applying the main phase of our PR approach, we fix all arcs that are already contained in both solutions. An arc (i, j), where i and j correspond to customers, is fixed by merging the vertex sequences $\Gamma^-_i = (y, ..., i)$ and $\Gamma^+_j = (j, ..., z)$. The result of the merging is stored in Γ^+_y and Γ^-_z, i.e., $\Gamma^+_y = \Gamma^-_z = (y, ..., i, j, ..., z)$. We then remove Γ^-_i and Γ^+_j from the set of fixed vertex sequences. Consider arc $(3, 11)$, that is contained in both example solutions to be fixed first. This requires merging of Γ^-_3 and Γ^+_{11}, leading to $\Gamma^+_3 = \Gamma^-_{11} = (3, 11)$. From now on, arc $(3, 11)$ is not allowed to be removed from S_c anymore. In other words, Γ^-_3 and Γ^+_{11} may not be removed from the fixed sequence they are now part of and are therefore removed from the sequence set. In case the regarded arc starts (ends) with the depot vertex 0, the corresponding depot visit is inserted at the beginning (end) of $\Gamma^+_j = (j, ..., z)$ $(\Gamma^-_i = (y, ..., i))$, leading to the modified sequence $\Gamma^-_z = (0, j, ..., z)$

($\Gamma_y^+ = (y, ..., i, 0)$) and discarding Γ_j^+ (Γ_i^-). For example, fixing arc $(0, 3)$ in Figure 4 after arc $(3, 11)$ results in $\Gamma_{11}^- = (0, 3, 11)$ and the removal of Γ_3^+.

Next, we successively incorporate the arcs from $\mathcal{A}^{\neq}$ into the current solution, thereby advancing on the solution path between S and S_g. Because the guiding solution is close to a local optimum, it is unlikely to discover improving solutions in its immediate proximity. Therefore, we refrain from exploring the entire path between both solutions and restrict the search to a fraction of ρ of the solution path. More precisely, we repeat the following steps until the number of remaining differing arcs is smaller than a fraction of $(1 - \rho)$ of the initial cardinality of $\mathcal{A}^{\neq}$:

For each arc in $\mathcal{A}^{\neq}$, we determine the best move that inserts this arc into the current solution. Consider arc (i, j) to be created in S_c. We distinguish five cases with respect to the creation of (i, j):

1. If Γ_i^- and Γ_j^+ do not contain the depot, two moves are possible: (i) Sequence Γ_i^- may be moved before Γ_j^+, or (ii) sequence Γ_j^+ can be relocated after sequence Γ_i^-. Assume arc $(7, 10)$ to be created in S_c in the first iteration of our example application. Then, we may relocate customer 7 from route E to route C, i.e., Γ_7^- before Γ_{10}^+. Alternatively, it is allowed to move customer 10 from route C to route E, i.e., Γ_{10}^+ after Γ_7^-.
2. If one of both sequences starts or ends with the depot, we only allow the relocation of the vertex sequence that is not connected to the depot. Sequence $\Gamma_1^- = (0, 1)$ in Figure 4, for instance, may not be relocated to create arc $(1, 8)$.
3. If Γ_i^- starts and Γ_j^+ ends with the depot and both sequences are part of the same route, arc (i, j) can only be implicitly created by relocating all vertices lying between i and j.
4. If Γ_i^- starts and Γ_j^+ ends with the depot and both sequences are served in different routes, the *2-opt** operator which is described in Section 2.5 is applied to merge the two route segments.
5. Finally, if the arc to be created starts (ends) with the depot, we determine the best relocation for Γ_j^+ (Γ_i^-) among the routes where the start (end) depot visit is not part of a fixed vertex sequence.

The overall best move out of the set of best moves per arc is subsequently applied to the current solution S_c. We set the best solution S' to S_c if (i) S_c does not correspond to the initial solution S or the guiding solution S_g, and (ii) S' has not been initialized yet, or S_c improves on S'.

Moreover, we remove the matching arcs created by the previously performed move from $\mathcal{A}^{\neq}$ and update the set of fixed vertex sequences accordingly. Note that each move might introduce more differing arcs than the currently considered arc into S_c. Take for example iteration 2 in Figure 4. Here, we identify the relocation of sequence $\Gamma_9^- = (9)$ from its former position in route D to the last position in route E as the best move to introduce arc $(9, 0)$ into S_c. In doing so, three differing arcs, namely $(1, 8)$, $(6, 9)$, and the regarded arc $(9, 0)$ are created at once and subsequently removed from $\mathcal{A}^{\neq}$. Next, the associated fixed vertex sequences need to be updated. Assume this is done according to the previously mentioned order of arcs. Then, we merge $\Gamma_1^- = (0, 1)$ and $\Gamma_8^+ = (8)$ into $\Gamma_8^- = (0, 1, 8)$, $\Gamma_6^- = (6)$ and $\Gamma_9^+ = (9)$ into $\Gamma_9^- = \Gamma_6^+ = (6, 9)$, and attach the depot to $\Gamma_9^- = (6, 9)$, leading to $\Gamma_6^+ = (6, 9, 0)$. Finally, Γ_1^-, Γ_8^+, Γ_6^-, Γ_9^+, and Γ_9^- are removed from the sequence set.

For the purpose of illustration, Figure 4 shows the creation of the entire solution path, leading to the equalization of both solutions in iteration 4.

2.5 Local search

In the initialization phase and after each application of our ALNS and PR components, a local search procedure is applied.

Iteration	Route	S_c	S_g	Arcs created in S_c
0	A	0 - 3 - 11 - 0	0 - 3 - 11 - 0	
	B	0 - 5 - 0	0 - 5 - 0	
	C	0 - 2 - 10 - 0	0 ✗ 7 ✗ 10 ✗ 4 ✗ 0	
	D	0 - 1 - 9 - 8 - 0	0 - 1 ✗ 8 ✗ 2 ✗ 0	
	E	0 - 4 - 7 - 6 - 0	0 ✗ 6 ✗ 9 ✗ 0	
1	A	0 - 3 - 11 - 0	0 - 3 - 11 - 0	
	B	0 - 5 - 0	0 - 5 - 0	
	C	0 - 2 - 7 - 10 - 0	0 ✗ 7 - 10 ✗ 4 ✗ 0	$(7, 10)$
	D	0 - 1 - 9 - 8 - 0	0 - 1 ✗ 8 ✗ 2 ✗ 0	
	E	0 - 4 - 6 - 0	0 ✗ 6 ✗ 9 ✗ 0	
2	A	0 - 3 - 11 - 0	0 - 3 - 11 - 0	
	B	0 - 5 - 0	0 - 5 - 0	
	C	0 - 2 - 7 - 10 - 0	0 ✗ 7 - 10 ✗ 4 ✗ 0	
	D	0 - 1 - 8 - 0	0 - 1 - 8 ✗ 2 ✗ 0	$(1, 8)$
	E	0 - 4 - 6 - 9 - 0	0 ✗ 6 - 9 - 0	$(6, 9), (9, 0)$
3	A	0 - 3 - 11 - 0	0 - 3 - 11 - 0	
	B	0 - 5 - 0	0 - 5 - 0	
	C	0 - 7 - 10 - 0	0 - 7 - 10 ✗ 4 ✗ 0	$(0, 7)$
	D	0 - 1 - 8 - 2 - 0	0 - 1 - 8 - 2 - 0	$(8, 2), (2, 0)$
	E	0 - 4 - 6 - 9 - 0	0 ✗ 6 - 9 - 0	
4	A	0 - 3 - 11 - 0	0 - 3 - 11 - 0	
	B	0 - 5 - 0	0 - 5 - 0	
	C	0 - 7 - 10 - 4 - 0	0 - 7 - 10 - 4 - 0	$(10, 4), (4, 0)$
	D	0 - 1 - 8 - 2 - 0	0 - 1 - 8 - 2 - 0	
	E	0 - 6 - 9 - 0	0 - 6 - 9 - 0	$(0, 6)$

Figure 4: Example application of our PR procedure

We aim at improving routes by means of the following operators: A *relocate* operator, originally introduced in Savelsbergh (1992) is implemented which moves a customer from its position to a different position in the same or a different route. Moreover, we use an *exchange* operator (Savelsbergh 1992) which is able to swap customers between and within routes. The *2-opt* operator replaces two arcs of a single route by two new ones reversing the order of customers between the vertices incident to the removed arcs (Lin 1965). Finally, we implement a *2-opt** operator as an inter-route modification of the 2-opt operator which removes one arc from each route and reconnects the first part of the first route with the second part of the second route and vice versa (Potvin and Rousseau 1995). A reversal of the route orientation, which is usually undesirable in problems with time windows, is hereby avoided.

Our local search implements a best improvement strategy. The search is stopped when no further improving move can be identified. In order to speed up the search, we maintain a set of changed routes which is initially filled with the routes that have been altered by the preceding algorithmic component. We restrict the evaluation and thus application of moves to those involving at least one route from this set. The set of changed routes is dynamically updated during the execution of the local search, i.e., a previously unchanged route is added to the set after being modified by a local search move.

2.6 Acceptance decision

To decide if the solution S' returned by the local search should replace S, we use an acceptance mechanism based on SA.

While solutions that improve S are always accepted, we accept worse solutions with probability $e^{-(f_{gen}(S')-f_{gen}(S))/T}$. The temperature parameter T is initially set to a value such that a deterioration of the current solution S by δ is accepted with a probability of 50%. The temperature is decreased after every ALNS-PR iteration such that the acceptance probability of a relative deterioration of δ is equal to 1% after a cooling period of τ iterations.

3 Computational studies

In this section, we present the computational studies conducted to evaluate the performance of our ALNS-PR. The benchmark instances used for testing our algorithm are presented in Section 3.1. Section 3.2 describes the experimental environment and the setting of the algorithmic parameters. In Section 3.3, we analyze the influence of specific components of our algorithm on solution quality and computation time. Finally, in Section 3.4, we discuss the competitiveness of our approach on VRPSPD, VRPSPDTL, VRPSPDTW, and VRPDDP benchmark instances from the literature, and present results for the newly introduced set of VRPDDPTW instances.

3.1 Benchmark instances

In the following, we describe the benchmark instances that we use in our computational studies structured according to problem type.

3.1.1 VRPSPD instances

To assess the performance of ALNS-PR for the VRPSPD, we use the instances proposed by Salhi and Nagy (1999), from now on referred to as set Salhi-VRPSPD, Dethloff (2001), denoted as set Dethloff, and Montané and Galvão (2006), for which the subset of instances with up to 200 customers is called Montané-Medium and the whole set Montané-All.

The set Salhi-VRPSPD is based on a CVRP benchmark introduced in Christofides et al. (1979) that contains 14 instances with 50 to 199 customers which are randomly distributed in ten instances and clustered in the remaining ones. Salhi and Nagy (1999) use seven of the CVRP istances that do not contain a maximum route duration and customer service times to generate seven VRPSPD instances by splitting the original customer demands into delivery and pickup demands based on a ratio of the customer coordinates. This subset of instances is referred to as X set. Another subset of seven instances (Y set) is generated from the X-series of instances by swapping the delivery and pickup demands of every other customer. Unfortunately, these instances are not consistently treated in the literature. Some authors swap the delivery and pickup demands of every customer instead of every other customer or round the distances and demand quantities. To be able to compare our algorithm to the state-of-the-art approaches for the VRPSPD, we do not round, but we exchange the demands of every customer in set X.

Set Dethloff contains 40 VRPSPD instances with 50 customers and three or eight vehicles. Half of the instances, denoted as SCA, are characterized by a uniform distribution of the customer coordinates. In the other 20 instances, denoted as CON, half of the customers are uniformly distributed and the remaining customers located in clusters.

The benchmark Montané-All contains 18 instances with $100, 200$, and 400 customers that are derived from a respective subset of the VRPTW instances proposed in Solomon (1987) and Gehring and Homberger (1999). The customers are clustered (prefix C), randomly distributed (prefix R), or a mixture of both (prefix RC). The authors create VRPSPD instances by omitting the time window constraints and assigning a discrete pickup demand to each customer that is randomly drawn from the interval used to generate the delivery demands in the original instances.

3.1.2 VRPSPDTL instances

VRPSPDTL instances are proposed by Salhi and Nagy (1999), called Salhi-VRPSPDTL, and Polat et al. (2015), which we from now on refer to as Polat-VRPSPDTL.

Set Salhi-VRPSPDTL is composed of 14 instances which are generated according to the same procedure as described in Section 3.1.1 to generate VRPSPD instances using the remaining seven CVRP instances of Christofides et al. (1979) which contain a maximum route duration and customer service times.

The Polat-VRPSPDTL instances are created by first converting the CVRP instances of Christofides and Eilon (1969) to VRPSPD instances according to the approach described in Nagy et al. (2015) and then adding time limits. The instances do not contain customer service times.

3.1.3 VRPSPDTW instances

Two sets of VRPSPDTW instances are provided by Wang and Chen (2012), which we denote as Wang-Medium, and Wang et al. (2015), from now on referred to as Wang-Large.

The instances of set Wang-Medium are created by adding pickup demands to the 56 VRPTW instances with 100 customers proposed by Solomon (1987). In addition to the customer distribution described in Section 3.1.1, each instance is either characterized by narrow time windows and small vehicle capacities (prefix C1/R1/RC1) or large time windows and large vehicle capacities (prefix C2/R2/RC2).

Wang-Large contains 30 large-scale instances with $200, 400, 600, 800$, and 1000 customers which are derived from a subset of the VRPTW instances introduced in Gehring and Homberger (1999) by adding pickup demands.

3.1.4 VRPDDP instances

For the VRPDDP, we use three instance sets of Nagy et al. (2015), which we label as Nagy1, Nagy2, and Nagy3 and two sets of instances proposed by Polat (2017), which we denote as Polat-VRPDDP1 and Polat-VRPDDP2. All VRPDDP instances are based on VRPSPD and VRPSPDTL instances and solved under the assumption of divisible customer demands.

The sets Nagy1, Nagy2, and Nagy3 are based on the 28 instances of Salhi-VRPSPD and Salhi-VRPSPDTL with distances and demands rounded to the nearest integer. The set Nagy1 corresponds to the unchanged original instances. Taking the original set as a basis, the delivery and pickup demands of each customer are increased by multiplying each value by four and adding ten percent of the vehicle capacity in order to obtain instance set Nagy2. The instances for benchmark Nagy3 are created by adding 75% of the vehicle capacity to the delivery demand and 20% of the vehicle capacity to the pickup demand of every odd customer and adding 20% of the vehicle capacity to the delivery demand and 75% of the vehicle capacity to the pickup demand of every even customer. Again, the resulting demand values are rounded to the nearest integer. Moreover, for the instances with a maximum duration imposed on the vehicle routes, we assume that the service time associated with a customer is incurred twice if it is served via two separate visits. Polat (2017) interprets these instances differently and considers only half of the original service time for each visit of a divided customer.

The instances of set Polat-VRPDDP1 are based on the VRPSPD benchmark Dethloff. Polat-VRPDDP2 corresponds to the instances of Montané-Medium.

3.1.5 VRPDDPTW instances

Finally, we generate a new set of VRPDDPTW instances, denoted as HS, by modifying the VRP-SPDTW instances contained in Wang-Medium similar to the approach described in Section 3.1.4.

More precisely, we add 20% of the vehicle capacity to the delivery demand and 5% of the vehicle capacity to the pickup demand of every odd customer and 5% of the vehicle capacity to the delivery demand and 20% of the vehicle capacity to the pickup demand of every even customer. Note that we do not round and that the resulting demand quantities are limited to the vehicle capacity. Again, the whole service time is incurred for each partial visit of a divided customer.

3.2 Computational environment and parameter setting

Our ALNS-PR is implemented as single-thread code in Java. All tests were performed on a Windows 10 Professional desktop computer with an Intel Core i5-6600 processor running at 3.30 GHz and 16 GB RAM. In all experiments, we performed ten runs on each instance.

To determine the parameter setting of our algorithm, the procedure proposed in Ropke and Pisinger (2006a) is adopted. Starting from a reasonably well-performing parameter setting found during the development of our method, we successively refine the setting of each parameter. To this end, we evaluate three values for each parameter. The best value is kept as the final setting for the respective parameter, and we subsequently proceed with tuning the next parameter.

The resulting parameter setting is shown in Table 1. With respect to the main ALNS-PR routine, we report the maximum number of iterations without improvement (ω), the number of non-improving iterations after which the current solution is reset to the current best (μ), the maximum number of unsuccessful vehicle reduction attempts (θ), the number of iterations during which the feasibility of the current solution could not be restored and whereupon an empty route is added (ι), and the number of iterations after which the weight update of the adaptive components is enabled (ν). To achieve a competitive solution quality and computation times on the different sets of benchmark instances, we set $\omega = 4000$ for all VRPSPD benchmarks and the sets Salhi-VRPSPDTL, Polat-VRPDDP1, and Polat-VRPDDP2. For the remaining benchmarks, we set $\omega = 500$.

For our ALNS component, we provide the boundaries of the removal interval ($\Psi^{||}$) and the distance threshold interval (Ψ^d), the scores to evaluate the performance of the adaptive components (σ^{best}, σ^{imp}, and σ^{acc}), the factor which controls how fast the adaptive weight adjustment reacts to the performance of the adaptive components (α), the number of iterations after which the weights of the adaptive components are updated (γ), the parameters to control the degree of randomness in specific removal operators (χ^{load}, χ^{nb}, χ^{worst}, and χ^{rel}), the weights used to calculate the relatedness measure between two customers (χ^c, χ^d, χ^p, and χ^e), the percentage of best insertions considered by the GRASP insertion operator (χ^{GRASP}), the factor κ to control the amount of diversification in operation evaluation measure Δf_{div}, and the interval from which a random number is drawn in the context of evaluation measure Δf_{noise} ($[\zeta_{min}, \zeta_{max}]$).

Regarding the PR component, λ denotes the size of the elite set, and ρ corresponds to the percentage of the solution path which we investigate in each PR execution.

We further report the initial (δ^0), minimum (δ^{min}), and maximum (δ^{max}) penalty factors, the penalty update factor (δ^{update}), and the numbers of ALNS-PR iterations after which the penalty factors are increased (η^+) as well as decreased (η^-).

Finally, the SA-inspired acceptance mechanism is characterized by a relative solution deterioration δ used to determine the initial and minimal temperature, the cooling period (τ), and the number of solution resets after which the temperature is reset to its initial value and the current solution is set to a solution randomly drawn from the elite set (ϵ).

ALNS-PR		**ALNS**		**PR**		**Penalties**		**SA**	
ω	4000/500	$\Psi^{\|\|}$	$[0.01, 0.25]$	λ	10	δ^0	10	δ	0.01
μ	500	Ψ^d	$(0.2, ..., 0.5)$	ρ	0.7	δ^{min}	0.1	τ	100
θ	200	σ^{best}, σ^{imp}, σ^{acc}	$6, 9, 3$			δ^{max}	10 000	ϵ	3
ι	100	α	0.2			δ^{update}	1.1		
ν	100	γ	20			η^+	2		
		χ^{load}	5			η^-	2		
		χ^{nb}	5						
		χ^{worst}	4						
		χ^{rel}	5						
		χ^c, χ^d, χ^p, χ^e	$2, 1, 1, 1$						
		χ^{GRASP}	0.3						
		κ	0.7						
		$[\zeta_{min}, \zeta_{max}]$	$[0.9, 1.1]$						

Table 1: Final parameter setting of our ALNS-PR.

3.3 Influence of algorithmic components

In this section, we analyze the effect of specific components of our ALNS-PR algorithm on solution quality and computation time. For this purpose, we use ten instances randomly drawn from the instance sets Salhi-VRPSPD and Salhi-VRPSPDTL.

We investigate the contribution of

1. specific ALNS components, namely the decomposition of the original problem into subsets of closely located routes, the route selection policies, the evaluation measures Δf_{div} and Δf_{noise}, the newly introduced load balance removal operator, and the random insertion operator, and
2. our PR implementation.

To this end, we consecutively disable each component while keeping the remaining components enabled. The corresponding results are compared to those obtained by our full ALNS-PR algorithm. More precisely, in Table 2, we contrast the average percentage gaps of the best solutions found to the BKS from the literature (Avg. Δ^b) of each algorithmic configuration. Furthermore, Δ^t denotes the percentage deviation of the average of the average run-times per instance of each configuration to the respective value of our full ALNS-PR (shown in bold).

For each reduced algorithmic configuration, we observe a decrease in solution quality compared to our complete approach with a maximum deviation of the best solution quality of 0.09%. Moreover, the large majority of variants also shows a significant increase in computation time if the respective component is omitted. This indicates that the associated components accelerate the convergence rate of the search by helping to discover promising solutions early.

If the current solution is not decomposed in the ALNS step, we observe a run-time increase of roughly 30%. Moreover, omitting the introduced route selection policies slows down computation by roughly 15%, on average. A negligible speed-up with a decrease in solution quality can be obtained

	Components													
Decomposition	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Route selection														
Cost	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Distance	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Efficiency	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
High avg. util.	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓
Low avg. util.	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
High max. load	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Low max. load	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓
Evaluation														
Δf_{noise}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓
Δf_{div}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
ALNS Operators														
Load balance removal	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
Random insertion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
Path relinking	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Avg. Δ^b(%)	**0.01**	0.03	0.05	0.05	0.10	0.03	0.03	0.02	0.03	0.02	0.02	0.03	0.03	0.04
Δ^t(%)		30.09	16.58	21.04	11.28	-0.32	25.85	13.18	15.60	36.75	-2.26	32.47	7.74	110.46

Table 2: Comparison of the performance of different algorithmic configurations.

when the high average utilization selection policy is not employed. Deactivating evaluation measure Δf_{noise} in the ALNS step seems to impede the search in finding good solutions early and thus to decelerate its convergence rate by over 36%. On the other hand, the omission of Δf_{div} results in a slight speed-up of roughly 2%. However, we prefer the increased solution quality by utilizing this component over the small decrease in computation time that can be observed if it is omitted. Furthermore, the newly introduced load balance removal operator seems to properly capture the characteristics of the VRPSPD. By deactivating the new operator, the resulting algorithmic configuration additionally takes roughly a third of the time spent by our full ALNS-PR algorithm. In addition, the results hint at an appropriate diversification behavior of the random insertion operator. Besides the increase in solution quality, its utilization results in a noticeable acceleration of the search.

Finally, the hybridization of our ALNS with the described PR implementation yields the most significant increase in convergence speed. Using only our ALNS algorithm more than doubles the computation time required compared to the proposed hybrid approach.

3.4 Computational results on benchmark instances

This section presents the results obtained by our ALNS-PR algorithm on the benchmark instances from the literature for the VRPSPD, the VRPSPDTL (Section 3.4.1), the VRPSPDTW (Section 3.4.2), and the VRPDDP (Section 3.4.3) and on the newly proposed VRPDDPTW instances (Section 3.4.4).

With respect to the benchmarks from the literature, we present aggregate views on the performance of all relevant heuristics. The corresponding detailed results can be found in Appendix A.

3.4.1 Results on VRPSPD and VRPSPDTL instances

In Table 3, we present the summarized results of all relevant heuristics on the VRPSPD (Salhi-VRPSPD, Dethloff, Montané-Medium, and Montané-All) and VRPSPDTL (Salhi-VRPSPDTL and Polat-VRPSPDTL) benchmark instances. The upper part of the table shows the state-of-the-art heuristics that have been applied to the VRPSPD but not the VRPSPDTL; the lower part of the table contains the most successful approaches that provide solutions for both problem types, including our ALNS-PR, which is the only method applied to each set of instances.

For each benchmark, we report the average percentage gap of the best solution quality based on several runs to the BKS (Avg. Δ^b) for each method that has been tested on the benchmark. The average percentage gap of the average solution quality to the BKS (Avg. Δ^a) is provided for those instance sets for which this measure is available from the large majority of comparison algorithms.

Moreover, we translate the run-times of all methods into a common time measure that takes into account the processors used. To this end, we relate the Passmark scores (see www.passmark.com) of the processors used in the computational studies of the papers to the score of our i5-6600. Each Passmark score is referring to the performance of a single core of the respective processor. In case of parallel solution approaches, we multiply the translated run-times by the number of utilized threads as reported in the corresponding paper. In addition, the run-times are multiplied by the number of runs performed by the respective algorithm. The resulting times in seconds are given as t^c. We are aware that, due to the use of different operating systems and programming languages, an exact run-time comparison is never possible. However, this procedure is the closest we can get to a fair comparison.

In the following, we list the algorithms that are compared in Table 3 and explain how to interpret the corresponding results and run-times:

- For **ZTK** (Zachariadis, Tarantilis, and Kiranoudis 2010) the number of runs performed to obtain the best solution reported is unknown and the run-time is based on the time elapsed when the best solution was found.
- The results of **SDBOF** (Subramanian, Drummond, Bentes, Ochi, and Farias 2010) are based on 50 runs performed by 256 parallel threads and the average time per run.
- For **GKA** (Goksal, Karaoglan, and Altiparmak 2013), **VCGP** (Vidal, Crainic, Gendreau, and Prins 2014), and **SUO** (Subramanian, Uchoa, and Ochi 2013), the table provides results based on ten runs and the average time per run.
- **P** (Polat 2017) uses six parallel threads for which the number of runs performed is unknown. The run-time is based on the average time required to obtain the best solution reported.
- For **PKKG** (Polat, Kalayci, Kulak, and Günther 2015), the results are based on ten runs and the time of the best run for benchmarks Salhi-VRPSPD and Salhi-VRPSPDTL, and on the average time per run for benchmark Polat-VRPSPDTL.
- The results reported for ALNS-PR are based on ten runs and the average time per run.

Our ALNS-PR belongs to the best-performing approaches showing an average gap of the best solutions to the BKS of 0.00% on the sets Salhi-VRPSPD, Dethloff, and Salhi-VRPSPDTL, and competitive run-times with regard to the most successful heuristics that allow a fair comparison. On set Montané-All, SUO is the only approach able to obtain the BKS for each instance while also spending by far the most computation time. The slightly worse solution quality achieved by ALNS-PR is accompanied by a run-time advantage of roughly 56% compared to SUO. The best trade-off between solution quality and run-time on these instances can be achieved by VCGP. On set Polat-VRPSPDTL, we improve three out of seven previous BKS (see Table A.5 in Appendix A) and note an average improvement of -0.15% while spending only a fraction of the run-time required by PKKG.

	ZTK		SDBOF			GKA		VCGP			P	
Benchmark	Avg. Δ^b(%)	t^c(s)	Avg. Δ^b(%)	Avg. Δ^a(%)	t^c(s)	Avg. Δ^b(%)	t^c(s)	Avg. Δ^b(%)	Avg. Δ^a(%)	t^c(s)	Avg. Δ^b(%)	t^c(s)
Salhi-VRPSPD	0.11	5.13 × ?	0.00	-	134 656	0.10	1294.87	0.00	-	518.60	-	-
Dethloff	0.00	0.91 × ?	0.00	-	18 048	0.00	19.60	-	-	-	0.00	18.90 × ?
Montané-Medium	0.15	7.48 × ?	0.02	0.10	232 064	-	-	0.03	0.10	827.30	0.04	811.02 × ?
Montané-All	0.47	26.79 × ?	0.17	0.30	1 156 352	-	-	0.07	0.20	2226.90	-	-

	SUO			PKKG			**ALNS-PR**		
Benchmark	Avg. Δ^b(%)	Avg. Δ^a(%)	t^c(s)	Avg. Δ^b(%)	Avg. Δ^a(%)	t^c(s)	Avg. Δ^b(%)	Avg. Δ^a(%)	t^c(s)
Salhi-VRPSPD	0.21	-	1710.40	0.02	-	337.50	0.00	-	1308.50
Dethloff	-	-	-	-	-	-	0.00	-	105.60
Montané-Medium	0.00	0.03	4943.60	-	-	-	0.03	0.23	2246.30
Montané-All	0.00	0.08	22 585.90	-	-	-	0.09	0.37	9872.40
Salhi-VRPSPDTL	0.00	0.14	629.30	0.18	0.19	1155.10	0.00	0.21	891.50
Polat-VRPSPDTL	-	-	-	0.00	0.14	1044.40	-0.15	0.16	52.10

Table 3: Comparison of the results of ALNS-PR to those of the state-of-the-art VRPSPD and VRPSPDTL heuristics from the literature on the Salhi-VRPSPD, Dethloff, Montané-Medium, Montané-All, Salhi-VRPSPDTL, and Polat-VRPSPDTL benchmark sets.

It is further noteworthy, that we are able to improve two previous BKS on set Salhi-VRPSPDTL during our entire testing activities (see Table A.4 in Appendix A). The robustness of our ALNS-PR is indicated by the small deviations of the average solution quality to the best solution quality.

3.4.2 Results on VRPSPDTW instances

Table 4 shows an aggregated view on the performance of ALNS-PR and **WMZS** (Wang, Mu, Zhao, and Sutherland 2015) on the VRPSPDTW benchmark sets Wang-Medium and Wang-Large. For WMZS, the results are based on 66 threads and the time of the best run. The total number of runs performed is unknown. In addition to the previously reported measures, we provide for each method and set of instances, the total absolute deviation of the number of employed vehicles in the best solution to the best-known vehicle number ($\sum \Delta^m$).

	WMZS			**ALNS-PR**		
Benchmark	$\sum \Delta^m$	Avg. Δ^b(%)	t^c(s)	$\sum \Delta^m$	Avg. Δ^b(%)	t^c(s)
Wang-Medium	0	1.12	3961.32 × ?	-17	-2.32	421.20
Wang-Large	0	0.00	170 326.86 × ?	-102	-16.93	19 076.30

Table 4: Comparison of the results of ALNS-PR to those of WMZS on the VRPSPDTW benchmark sets Wang-Medium and Wang-Large.

On benchmark Wang-Medium, our ALNS-PR is able to reduce the number of employed vehicles for 17 out of 56 instances. On each of the remaining instances, we match the best-known vehicle number and significantly reduce the associated traveled distance for 31 instances (see Table A.6 in Appendix A). The average gaps of the best solution quality to the BKS are calculated across all instances where the reported vehicle number is equal to the previous best-known number of vehicles. For ALNS-PR, we note an average improvement of the previous BKS of -2.32%.

On set Wang-Large, we reduce the number of vehicles for 22 out of 30 instances by an average count of five vehicles. For the remaining instances, we match the previous best-known vehicle number and significantly reduce the associated traveled distance. In contrast to Wang-Medium, we now consider all instances for calculating the average gap of the best solution quality to the BKS, i.e., even instances for which we obtain a smaller number of vehicles. In general, a reduction of the number of employed vehicles may increase the traveled distance. However, as shown in Table A.7 in Appendix A, for each except one instance where we employ fewer vehicles, we also achieve a notable reduction of the previous best traveled distance as reported by WMZS. On average, we observe a gap of the best solution quality to the BKS of -16.93%.

Finally, it is remarkable that on both benchmark sets, ALNS-PR spends in total only a fraction of the time required by a single run of algorithm WMZS.

3.4.3 Results on VRPDDP instances

In this section, we analyze the performance of our ALNS-PR on the VRPDDP benchmarks Nagy1, Nagy2, Nagy3, Polat-VRPDDP1, and Polat-VRPDDP2. To this end, we compare our results on these instances to those of the algorithms **NWSA** (Nagy, Wassan, Speranza, and Archetti 2015) and P (cp. Section 3.4.1).

With respect to NWSA, the authors initially conducted experiments using a reactive tabu search algorithm that first determines a VRPSPD solution, which is then transformed into a VRPMDP

	NWSA		P		ALNS-PR		
Benchmark	Avg. Δ^b(%)	t^c(s)	Avg. Δ^b(%)	t^c(s)	$\sum n^d$	Avg. Δ^b(%)	t^c(s)
Nagy1	3.98	11.99	0.00	1248.40 × ?	36	-1.42	586.50
Nagy2	2.86	10.25	0.00	2500.10 × ?	865	-0.40	490.40
Nagy3	-	-	0.00	2848.00 × ?	1743	-0.61	351.30
Polat-VRPDDP1	-	-	0.00	73.50 × ?	23	-0.06	348.20
Polat-VRPDDP2	-	-	0.01	3037.98 × ?	12	-0.03	5378.30

Table 5: Comparison of the results of ALNS-PR to those of the heuristics of NWSA and P on the VRPDDP benchmark sets Nagy1, Nagy2, Nagy3, Polat-VRPDDP1, and Polat-VRPDDP2.

solution with twice as many customers (cp. Section 2). Subsequently, the authors develop several algorithmic variants that duplicate not all but only those customers identified to be promising for division according to the insights gained from the previous experiments. The following comparison with NWSA is based on the algorithmic version called DVA, which additionally makes use of an operator that allows for dividing customers during the execution of the algorithm and showed the best performance on average. The results reported for NWSA are based on a single run and the associated run-time. Unfortunately, we could not obtain a Passmark score for the processor used by NWSA (UltraSPARC-IIIi). Following P, we use an equivalent Intel Pentium 4 processor running at 1.90 GHz (Passmark score: 209) to translate the run-times reported by NWSA.

Table 5 shows the aggregated results of NWSA, P, and ALNS-PR on the different VRPDDP instance sets. P and ALNS-PR provide solutions for each set of VRPDDP benchmark instances; NWSA has only been applied to the sets Nagy1 and Nagy2. Therefore, we do not report results for NWSA on set Nagy3. For ALNS-PR, we additionally report the total number of customers that have been divided in the best solutions on each benchmark ($\sum n^d$). Unfortunately, this measure is neither reported for NWSA nor P.

On the instances of sets Nagy1, Nagy2, and Nagy3, algorithm P is able to match or significantly improve each previous BKS reported by the authors NWSA. Our ALNS-PR further improves the results of P by -1.42% (Nagy1), -0.40% (Nagy2), and -0.61% (Nagy3) on average while showing notably faster computation times. More precisely, the total time spent by ALNS-PR is significantly lower than the time required to obtain the best solution by a single run of method P on each of the three benchmarks. This is even more remarkable taking into account our more restrictive interpretation of service times in case of divided customers on these instances. As shown in Table A.8 in Appendix A, our ALNS-PR is able to improve the previous BKS for 20 instances in Nagy1, for 20 instances in Nagy2, and for 26 instances in Nagy3 out of 28 instances each. Moreover, only roughly 1% of all customers are divided on set Nagy1, 27% on Nagy2 and roughly 55% on Nagy3. Interestingly, NWSA report a total of 61 divided customers on Nagy1 for their basic approach while only 36 customers are identified by ALNS-PR to be divided. This might hint at a better capability of our ALNS-PR to identify and reassemble partial customers that have been unnecessarily divided in the course of the search.

Our ALNS-PR is superior to P also on the recently introduced VRPDDP benchmarks Polat-VRPDDP1 and Polat-VRPDDP2. We improve the previous best results by -0.06% and -0.03% on average, and obtain seven and two new BKS, respectively (see Tables A.9 and A.10 in Appendix A). Moreover, from a single-run perspective, ALNS-PR is roughly twice as fast as P on set Polat-VRPDDP1 and nearly six times as fast on set Polat-VRPDDP2.

3.4.4 Results on VRPDDPTW instances

In Table 6, we finally provide detailed results on the newly generated VRPDDPTW instances as comparison for future methods that address the VRPDDPTW. Moreover, in order to analyze the savings achievable by dividing customer demands, we compare the VRPDDPTW solutions to VRPSPDTW solutions obtained by our ALNS-PR on the new instances. We provide for each instance, the name and the number of customers, and for both problem types, the number of employed vehicles (m) and the best solution found in ten runs (f^b). For the VRPDDPTW, we additionally report the absolute deviation of the number of employed vehicles in the best solution to the vehicle number in the corresponding best VRPSPDTW solution, the number of divided customers in the best solution (n^d), the percentage gap of the best solution to the best VRPSPDTW solution (Δ^b), and the average computing time (t^a) in seconds.

Out of 56 instances, our ALNS-PR identifies 38 instances for which the division of demands is beneficial. In total, 652 customers are divided, allowing for a reduction of the number of employed vehicles by 23 compared to the VRPSPDTW case. On average, the traveled distance on the instances with an equal number of vehicles can be improved by -1.83% if dividing demands is permitted. Finally, our ALNS-PR shows reasonable computation times of less than one minute on average.

4 Conclusion

We present an adaptive large neighborhood search algorithm combined with a path relinking approach, called ALNS-PR, to address a class of VRPs with simultaneous pickup and delivery (VRPSPD).

In extensive numerical studies, we first demonstrate the usefulness of the proposed algorithmic components. We show that the omission of each component leads to a decrease in solution quality. Moreover, we find that especially the hybridization of our ALNS component with the proposed PR implementation, and the introduction of an innovative ALNS operator, which explicitly considers the load characteristics of the VRPSPD and its variants, significantly accelerates the convergence rate of the search.

The competitiveness of the proposed approach is demonstrated on benchmark instances from the literature. On established instances, ALNS-PR can compete with the state-of-the-art approaches for the corresponding problems. With respect to the more recently introduced problem variants, especially on VRPSPDTL, VRPSPDTW, and VRPDDP instances, our method proves to be superior to the majority of comparison algorithms and provides numerous new best solutions.

Our ALNS-PR is therefore suitable to tackle the practical application occurring at DHL Freight in Sweden that particularly motivated this paper. There, up to 50 000 customers need to be served from multiple depots. The scale of the resulting multi-depot VRPSPD suggests its decomposition into several VRPSPDs that can be solved independently. Future research thus needs to address the necessary algorithmic modifications to appropriately decompose a corresponding large-scale instance.

		ALNS-PR							
		VRPSPDTW		**VRPDDPTW**					
Inst.	n	m	f^b	m	Δ^m	n^d	f^b	Δ^b(%)	t^a(s)
HS-cdp101	100	23	**2345.75**	23	0	0	**2345.75**	0.00	27.04
HS-cdp102	100	23	**1840.89**	23	0	0	**1840.89**	0.00	31.10
HS-cdp103	100	22	1799.17	22	0	17	**1761.04**	-2.12	30.44
HS-cdp104	100	22	1752.16	22	0	8	**1652.09**	-5.71	34.93
HS-cdp105	100	23	**2085.45**	23	0	0	**2085.45**	0.00	22.86
HS-cdp106	100	22	**2772.87**	22	0	0	**2772.87**	0.00	27.39
HS-cdp107	100	22	**2684.66**	22	0	0	**2684.66**	0.00	25.92
HS-cdp108	100	22	**2378.12**	22	0	0	**2378.12**	0.00	25.79
HS-cdp109	100	22	**1982.66**	22	0	0	**1982.66**	0.00	29.35
HS-cdp201	100	17	1611.83	17	0	7	**1486.13**	-7.80	66.93
HS-cdp202	100	17	1461.67	16	**-1**	13	1803.94	23.42	52.97
HS-cdp203	100	17	1422.22	16	**-1**	13	1411.96	-0.72	71.46
HS-cdp204	100	17	1394.54	16	**-1**	12	1385.99	-0.61	60.97
HS-cdp205	100	17	1433.86	16	**-1**	26	2202.92	53.64	73.98
HS-cdp206	100	17	1421.85	16	**-1**	24	2102.17	47.85	50.61
HS-cdp207	100	17	1409.30	17	0	6	**1396.02**	-0.94	57.86
HS-cdp208	100	17	1421.41	16	**-1**	17	2184.17	53.66	64.15
HS-rdp101	100	22	**1965.84**	22	0	0	**1965.84**	0.00	25.92
HS-rdp102	100	21	**2031.76**	21	0	0	**2031.76**	0.00	36.01
HS-rdp103	100	21	**1662.40**	21	0	0	**1662.40**	0.00	35.77
HS-rdp104	100	20	**1906.62**	20	0	0	**1906.62**	0.00	42.75
HS-rdp105	100	21	**1967.15**	21	0	0	**1967.15**	0.00	26.10
HS-rdp106	100	20	**2467.94**	20	0	0	**2467.94**	0.00	39.30
HS-rdp107	100	20	1962.77	20	0	27	**1925.47**	-1.90	38.56
HS-rdp108	100	20	1623.36	20	0	4	**1608.53**	-0.91	36.84
HS-rdp109	100	21	**1668.95**	21	0	0	**1668.95**	0.00	31.83
HS-rdp110	100	20	**2239.71**	20	0	0	**2239.71**	0.00	35.61
HS-rdp111	100	21	**1637.16**	21	0	0	**1637.16**	0.00	33.30
HS-rdp112	100	20	1657.49	20	0	0	**1656.78**	-0.04	32.43
HS-rdp201	100	15	2337.05	15	0	10	**2090.14**	-10.57	52.64
HS-rdp202	100	15	1572.11	14	**-1**	39	1936.52	23.18	105.56
HS-rdp203	100	15	1635.94	14	**-1**	28	2256.06	37.91	113.51
HS-rdp204	100	15	1314.99	14	**-1**	17	1675.77	27.44	108.30
HS-rdp205	100	15	1671.05	14	**-1**	43	2202.27	31.79	92.56
HS-rdp206	100	15	1547.44	14	**-1**	31	1854.27	19.83	94.33
HS-rdp207	100	15	1350.41	14	**-1**	34	1678.72	24.31	98.72
HS-rdp208	100	15	1333.82	14	**-1**	31	1415.20	6.10	109.50
HS-rdp209	100	15	1609.23	14	**-1**	22	2069.46	28.60	79.36
HS-rdp210	100	15	1544.21	14	**-1**	41	1965.03	27.25	116.10
HS-rdp211	100	15	1365.45	15	0	11	**1209.21**	-11.44	117.33
HS-rcdp101	100	23	2340.21	23	0	1	**2336.11**	-0.18	23.69
HS-rcdp102	100	22	2435.76	22	0	19	**2320.66**	-4.73	32.72
HS-rcdp103	100	22	2173.06	22	0	10	**2038.64**	-6.19	45.59
HS-rcdp104	100	22	2011.60	22	0	12	**1967.52**	-2.19	56.28
HS-rcdp105	100	22	**2588.82**	22	0	0	**2588.82**	0.00	22.42
HS-rcdp106	100	22	2386.96	22	0	1	**2374.35**	-0.53	28.48
HS-rcdp107	100	22	2121.41	22	0	11	**2033.68**	-4.14	36.16
HS-rcdp108	100	22	2021.30	22	0	7	**1999.16**	-1.10	37.99
HS-rcdp201	100	16	2004.33	15	**-1**	17	2859.48	42.67	72.38
HS-rcdp202	100	16	1716.64	15	**-1**	12	1672.16	-2.59	82.14
HS-rcdp203	100	16	1588.73	15	**-1**	17	1518.74	-4.41	64.54
HS-rcdp204	100	16	1584.29	15	**-1**	10	1464.00	-7.59	91.59
HS-rcdp205	100	16	1874.17	15	**-1**	8	2239.42	19.49	60.44
HS-rcdp206	100	16	1693.38	15	**-1**	16	1973.35	16.53	78.13
HS-rcdp207	100	16	1689.49	15	**-1**	17	1602.60	-5.14	80.79
HS-rcdp208	100	16	1572.66	15	**-1**	13	1445.75	-8.07	82.49
Avg.								**-1.83**	**56.28**
$\sum$					**-23**	**652**			

Table 6: Results on the newly generated VRPDDPTW benchmark HS.

References

O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

N. Christofides and S. Eilon. An algorithm for the vehicle-dispatching problem. *Operational Research Quarterly*, 20(3):309–318, 1969.

N. Christofides, A. Mingozzi, and P. Toth. *The vehicle routing problem.* 1979.

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

J. Dethloff. Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up. *OR Spectrum*, 23(1):79–96, 2001.

A. E. Fallahi, C. Prins, and R. W. Calvo. A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem. *Computers & Operations Research*, 35(5):1725–1741, 2008.

T. A. Feo and M. G. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

H. Gehring and J. Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In K. Miettinen, M. Mkel, and J. Toivanen, editors, *Proceedings of EUROGEN99*, number A2, pages 57–64. Springer, Berlin, 1999.

F. Glover. *Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges*, pages 1–75. Springer US, Boston, MA, 1997.

D. Goeke and M. Schneider. Routing a mixed fleet of electric and conventional vehicles. *European Journal of Operational Research*, 245(1):81 – 99, 2015.

F. P. Goksal, I. Karaoglan, and F. Altiparmak. A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. *Computers & Industrial Engineering*, 65(1):39–53, 2013.

V. C. Hemmelmayr, J.-F. Cordeau, and T. G. Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39(12): 3215–3228, 2012.

S. C. Ho and M. Gendreau. Path relinking for the vehicle routing problem. *Journal of Heuristics*, 12(1):55–72, 2006.

J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7, 1956.

S. Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10): 2245–2269, 1965.

F. A. T. Montané and R. D. Galvão. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers & Operations Research*, 33(3):595–619, 2006.

Y. Nagata, O. Bräysy, and W. Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 37(4):724–737, 2010.

G. Nagy, N. A. Wassan, M. G. Speranza, and C. Archetti. The vehicle routing problem with divisible deliveries and pickups. *Transportation Science*, 49(2):271–294, 2015.

V.-P. Nguyen, C. Prins, and C. Prodhon. Solving the two-echelon location routing problem by a GRASP reinforced by a learning process and path relinking. *European Journal of Operational Research*, 216(1): 113–126, 2012.

D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

O. Polat. A parallel variable neighborhood search for the vehicle routing problem with divisible deliveries and pickups. *Computers & Operations Research*, 85:71–86, 2017.

O. Polat, C. B. Kalayci, O. Kulak, and H.-O. Günther. A perturbation based variable neighborhood search heuristic for solving the vehicle routing problem with simultaneous pickup and delivery with time limit. *European Journal of Operational Research*, 242(2):369–382, 2015.

J. Potvin and J. Rousseau. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446, 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.

S. Salhi and G. Nagy. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *The Journal of the Operational Research Society*, 50(10):1034–1042, 1999.

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

M. Schneider, B. Sand, and A. Stenger. A note on the time travel approach for handling time windows in vehicle routing problems. *Computers & Operations Research*, 40(10):2564–2568, 2013.

M. Schneider, A. Stenger, and J. Hof. An adaptive VNS algorithm for vehicle routing problems with intermediate stops. *OR Spectrum*, 37(2):353–387, 2015.

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

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg, 1998.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

J. R. Stock. Reverse logistics: white paper. *Council of Logistics Management*, 1992.

A. Subramanian, L. M. A. Drummond, C. Bentes, L. S. Ochi, and R. Farias. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 37 (11):1899–1911, 2010.

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.

T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673, 2014.

C. Wang, D. Mu, F. Zhao, and J. W. Sutherland. A parallel simulated annealing method for the vehicle routing problem with simultaneous pickupdelivery and time windows. *Computers & Industrial Engineering*, 83 (Complete):111–122, 2015.

H.-F. Wang and Y.-Y. Chen. A genetic algorithm for the simultaneous delivery and pickup problems with time window. *Computers & Industrial Engineering*, 62(1):84–95, 2012.

E. E. Zachariadis, C. D. Tarantilis, and C. T. Kiranoudis. An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries. *European Journal of Operational Research*, 202(2):401–411, 2010.

A Detailed results on the benchmark instances from the literature

In the following, we present the detailed results on the benchmark instances from the literature structured according to problem type.

VRPSPD Tables A.1, A.2, and A.3 show detailed comparisons of our ALNS-PR with the state-of-the-art approaches for the VRPSPD on the benchmarks Salhi-VRPSPD, Dethloff, and Montané-All (including Montané-Medium), respectively.

We report for each instance, the name, the number of customers (n), and the BKS from the literature. For each method, we provide the percentage gap of the best solution found in several runs to the BKS (Δ^b) and the average computing time (t^a), the time elapsed when the best solution was found (t^b), the average time elapsed when the best solution was found (t^{ab}), or the total time of the best run (t) in seconds (see Section 3.4 for details on how to interpret the results of the comparison algorithms). In Table A.3, the percentage gap of the average solution quality to the BKS (Δ^a) is additionally given. Values in bold indicate the best solution quality for each instance. Averages of the gaps to the BKS and the run-times are reported after the detailed results per instance at the end of each table.

VRPSPDTL In Tables A.4 and A.5, we present the detailed results on the VRPSPDTL benchmarks Salhi-VRPSPDTL and Polat-VRPSPDTL, respectively.

In the course of our computational experiments, we were able to obtain two new BKS on the instances of Salhi-VRPSPDTL. In addition to the previously reported measures, we therefore report in Table A.4, the best solutions encountered during our entire testing activities (f) and the corresponding percentage gaps to the previous BKS (Δ) in column $\overline{\text{ALNS-PR}}$.

On set Polat-VRPSPDTL, we improve three out of seven previous BKS based on ten runs. The corresponding absolute solution values are given in column f^b of Table A.5.

VRPSPDTW Tables A.6 and A.7 show the detailed results on the VRPSPDTW instance sets Wang-Medium and Wang-Large, respectively.

In addition to the previous measures, we report the number of employed vehicles in the best solution (m) and the absolute deviation of the number of vehicles in the best solution found to the previous best-known vehicle number (Δ^m) for each instance and solution method.

With respect to Table A.7, note that we identified the WMZS solutions on instances RC2_8_1 and RC2_10_1 to be infeasible (indicated by asterisks). The average route lengths of 1789.81 and 2119.78 calculated for both solutions exceed the latest arrival times at the depot given in the corresponding instances of 1573 and 1821, respectively. Therefore, we omit those instances when analyzing the solution quality of both comparison algorithms.

VRPDDP Finally, we report our detailed results for the VRPDDP on the benchmarks Nagy1, Nagy2, and Nagy3 in Table A.8 and on the benchmarks Polat-VRPDDP1 and Polat-VRPDDP2 in Tables A.9 and A.10, respectively. We report the same measures as in the previous tables, and additionally for ALNS-PR, the number of customers that have been divided in the best solution found (n^d) for each instance.

			ZTK		SDBOF		GKA		SUO		VCGP		PKKG		**ALNS-PR**	
Inst.	n	BKS	Δ^b(%)	t^b(s)	Δ^b(%)	t^a(s)	Δ^b(%)	t^a(s)	Δ^b(%)	t^a(s)	Δ^b(%)	t^a(s)	Δ^b(%)	t(s)	Δ^b(%)	t^a(s)
CMT1X	50	466.77	0.65	2.10	**0.00**	2.28	**0.00**	1.30	**0.00**	2.08	**0.00**	43.20	**0.00**	16.52	**0.00**	13.31
CMT1Y	50	466.77	0.65	3.80	**0.00**	2.27	**0.00**	1.40	**0.00**	1.97	**0.00**	42.60	**0.00**	8.26	**0.00**	12.20
CMT2X	75	684.21	**0.00**	5.40	**0.00**	6.44	**0.00**	35.60	**0.00**	12.79	**0.00**	79.20	**0.00**	44.92	**0.00**	32.03
CMT2Y	75	684.21	**0.00**	6.80	**0.00**	6.41	**0.00**	36.80	**0.00**	10.83	**0.00**	81.00	**0.00**	46.73	**0.00**	34.90
CMT3X	100	721.27	**0.00**	11.90	**0.00**	12.10	**0.00**	41.70	**0.00**	17.69	**0.00**	101.40	**0.00**	52.18	**0.00**	72.55
CMT3Y	100	721.27	**0.00**	11.00	**0.00**	12.28	**0.00**	55.50	**0.00**	17.61	**0.00**	107.40	**0.00**	46.09	**0.00**	127.43
CMT12X	100	662.22	**0.00**	9.30	**0.00**	10.29	0.11	141.40	**0.00**	9.07	**0.00**	104.40	**0.00**	33.91	**0.00**	39.97
CMT12Y	100	662.22	**0.00**	4.80	**0.00**	10.76	0.19	105.40	**0.00**	9.34	**0.00**	101.40	**0.00**	33.34	**0.00**	41.82
CMT11X	120	833.92	**0.00**	21.20	**0.00**	18.87	**0.00**	244.90	1.48	51.82	**0.00**	165.00	**0.00**	33.91	**0.00**	99.29
CMT11Y	120	833.92	**0.00**	14.40	**0.00**	19.03	**0.00**	368.90	1.48	48.63	**0.00**	159.60	**0.00**	49.64	**0.00**	98.83
CMT4X	150	852.46	**0.00**	29.60	**0.00**	30.89	0.04	380.20	**0.00**	98.03	**0.00**	249.60	**0.00**	118.97	**0.00**	187.30
CMT4Y	150	852.46	**0.00**	27.40	**0.00**	31.61	**0.00**	414.60	**0.00**	80.63	**0.00**	237.00	**0.00**	136.37	**0.00**	216.23
CMT5X	199	1029.25	0.13	62.80	**0.00**	71.50	0.41	500.00	**0.00**	1786.74	**0.00**	479.40	0.13	554.39	0.06	407.47
CMT5Y	199	1029.25	0.13	47.70	**0.00**	69.58	0.66	500.00	**0.00**	1726.18	**0.00**	396.00	0.13	287.05	**0.00**	448.52
Avg.			**0.11**	**18.44**	**0.00**	**21.74**	**0.10**	**201.98**	**0.21**	**276.67**	**0.00**	**167.66**	**0.02**	**104.45**	**0.00**	**130.85**
Processor type			T5500		Xeon		Xeon		i7		Opteron 250		Core 2 Duo T5750		i5-6600	
Processor speed			1.66 GHz		2.66 GHz		3.16 GHz		2.93 GHz		2.4 GHz		2.00 GHz		3.30 GHz	
Passmark score			584		1015		1345		1297		649		678		2098	
t^c(s)			**5.13 × ?**		**10.52 × 256 × 50**		**129.49 × 10**		**171.04 × 10**		**51.86 × 10**		**33.75 × 10**		**130.85 × 10**	

Table A.1: Detailed results of ALNS-PR and the state-of-the-art heuristics for the VRPSPD on the Salhi-VRPSPD benchmark.

			ZTK		SDBOF		GKA		P		**ALNS-PR**	
Inst.	n	BKS	Δ^b(%)	t^b(s)	Δ^b(%)	t^a(s)	Δ^b(%)	t^a(s)	Δ^b(%)	t^{ab}(s)	Δ^b(%)	t^a(s)
SCA3-0	50	635.62	0.00	2.50	0.00	2.31	0.00	4.90	0.00	4.77	0.00	19.37
SCA3-1	50	697.84	0.00	2.50	0.00	2.28	0.00	0.80	0.00	5.76	0.00	14.59
SCA3-2	50	659.34	0.00	2.90	0.00	2.14	0.00	0.40	0.00	8.22	0.00	22.38
SCA3-3	50	680.04	0.00	2.30	0.00	2.49	0.00	1.00	0.00	6.24	0.00	11.74
SCA3-4	50	690.50	0.00	2.90	0.00	2.18	0.00	0.30	0.00	3.97	0.00	11.95
SCA3-5	50	659.90	0.00	3.00	0.00	2.23	0.00	2.00	0.00	5.70	0.00	16.41
SCA3-6	50	651.09	0.00	3.10	0.00	2.51	0.00	0.80	0.00	5.15	0.00	17.31
SCA3-7	50	659.17	0.00	2.80	0.00	2.49	0.00	1.60	0.00	6.67	0.00	12.38
SCA3-8	50	719.47	0.00	3.50	0.00	2.26	0.00	0.50	0.00	4.96	0.00	12.73
SCA3-9	50	681.00	0.00	4.70	0.00	1.90	0.00	0.80	0.00	8.50	0.00	15.45
SCA8-0	50	961.50	0.00	2.70	0.00	3.37	0.00	4.80	0.00	5.33	0.00	11.69
SCA8-1	50	1049.65	0.00	3.80	0.00	2.89	0.00	6.80	0.00	6.74	0.00	7.33
SCA8-2	50	1039.64	0.00	3.90	0.00	2.38	0.00	10.20	0.00	5.45	0.00	9.42
SCA8-3	50	983.34	0.00	2.60	0.00	2.98	0.00	13.00	0.00	8.39	0.00	9.24
SCA8-4	50	1065.49	0.00	2.40	0.00	2.81	0.00	3.00	0.00	6.07	0.00	7.96
SCA8-5	50	1027.08	0.00	3.40	0.00	3.31	0.00	4.10	0.00	5.57	0.00	13.51
SCA8-6	50	971.82	0.00	2.70	0.00	3.51	0.00	1.60	0.00	6.98	0.00	10.09
SCA8-7	50	1051.28	0.00	5.10	0.00	3.12	0.00	3.40	0.00	9.77	0.00	10.38
SCA8-8	50	1071.18	0.00	3.60	0.00	2.92	0.00	0.80	0.00	7.06	0.00	10.21
SCA8-9	50	1060.50	0.00	4.80	0.00	2.18	0.00	7.30	0.00	5.82	0.00	9.09
CON3-0	50	616.52	0.00	4.70	0.00	3.12	0.00	2.10	0.00	6.12	0.00	8.21
CON3-1	50	554.47	0.00	2.20	0.00	2.83	0.00	1.30	0.00	4.01	0.00	14.49
CON3-2	50	518.00	0.00	3.10	0.00	2.77	0.00	1.30	0.00	9.06	0.00	10.42
CON3-3	50	591.19	0.00	3.20	0.00	2.34	0.00	0.50	0.00	6.90	0.00	22.12
CON3-4	50	588.79	0.00	2.30	0.00	2.63	0.00	3.20	0.00	3.12	0.00	9.67
CON3-5	50	563.70	0.00	3.70	0.00	2.69	0.00	0.40	0.00	6.17	0.00	8.95
CON3-6	50	499.05	0.00	3.70	0.00	2.75	0.00	2.30	0.00	9.39	0.00	10.77
CON3-7	50	576.48	0.00	1.90	0.00	2.75	0.00	2.60	0.00	4.69	0.00	11.05
CON3-8	50	523.05	0.00	3.80	0.00	2.46	0.00	1.00	0.00	3.89	0.00	8.03
CON3-9	50	578.25	0.00	2.20	0.00	3.37	0.00	2.90	0.00	5.70	0.00	8.02
CON8-0	50	857.17	0.00	4.40	0.00	3.65	0.00	5.20	0.00	4.86	0.00	5.21
CON8-1	50	740.85	0.00	3.30	0.00	3.02	0.00	2.90	0.00	6.77	0.00	5.69
CON8-2	50	712.89	0.00	2.70	0.00	3.08	0.00	2.10	0.00	3.83	0.00	9.50
CON8-3	50	811.07	0.00	2.80	0.00	3.99	0.00	2.80	0.00	6.49	0.00	4.69
CON8-4	50	772.25	0.00	2.80	0.00	3.69	0.00	3.60	0.00	6.70	0.00	7.40
CON8-5	50	754.88	0.00	5.70	0.00	4.18	0.00	3.40	0.00	6.31	0.00	5.84
CON8-6	50	678.92	0.00	3.40	0.00	4.09	0.00	7.90	0.00	4.80	0.00	5.33
CON8-7	50	811.96	0.00	2.50	0.00	4.03	0.00	3.00	0.00	9.22	0.00	4.12
CON8-8	50	767.53	0.00	3.20	0.00	3.42	0.00	3.20	0.00	4.93	0.00	4.98
CON8-9	50	809.00	0.00	3.80	0.00	3.48	0.00	2.40	0.00	6.47	0.00	4.81
Avg.			**0.00**	**3.27**	**0.00**	**2.92**	**0.00**	**3.06**	**0.00**	**6.16**	**0.00**	**10.56**
Processor type			T5500		Xeon		Xeon		Xeon E5420		i5-6600	
Processor speed			1.66 GHz		2.66 GHz		3.16 GHz		2.50 GHz		3.30 GHz	
Passmark score			584		1015		1345		1073		2098	
t^c(s)			**0.91 × ?**		**1.41 × 256 × 50**		**1.96 × 10**		**3.15 × 6 × ?**		**10.56 × 10**	

Table A.2: Detailed results of ALNS-PR and the state-of-the-art heuristics for the VRPSPD on the Dethloff benchmark.

			ZTK		SDBOF			SUO			VCGP			P		**ALNS-PR**		
Inst.	n	BKS	Δ^b(%)	t^b(s)	Δ^b(%)	Δ^a(%)	t^a(s)	Δ^b(%)	Δ^a(%)	t^a(s)	Δ^b(%)	Δ^a(%)	t^a(s)	Δ^b(%)	t^{ab}(s)	Δ^b(%)	Δ^a(%)	t^a(s)
r101	100	1009.95	**0.00**	5.80	**0.00**	0.06	15.81	**0.00**	0.01	65.42	**0.00**	0.16	73.80	**0.00**	33.86	**0.00**	0.49	46.61
r201	100	666.20	**0.00**	7.90	**0.00**	**0.00**	15.95	**0.00**	**0.00**	15.71	**0.00**	**0.00**	143.40	**0.00**	30.56	**0.00**	0.33	134.67
c101	100	1220.18	0.07	8.60	**0.00**	0.04	10.39	**0.00**	0.02	12.93	0.07	0.07	66.00	**0.00**	15.63	0.07	0.25	52.62
c201	100	662.07	**0.00**	4.30	**0.00**	**0.00**	8.83	**0.00**	**0.00**	9.77	**0.00**	**0.00**	97.80	**0.00**	15.55	**0.00**	**0.00**	58.92
rc101	100	1059.32	**0.00**	14.10	**0.00**	**0.00**	11.07	**0.00**	**0.00**	16.89	**0.00**	**0.00**	72.60	**0.00**	18.25	**0.00**	**0.00**	40.09
rc201	100	672.92	**0.00**	10.60	**0.00**	**0.00**	7.28	**0.00**	**0.00**	11.42	**0.00**	**0.00**	119.40	**0.00**	18.86	**0.00**	0.03	85.07
R1_2_1	200	3353.80	0.67	45.10	0.19	0.48	66.21	**0.00**	0.04	1142.05	0.05	0.32	406.20	0.19	370.83	0.17	0.59	243.15
R2_2_1	200	1665.58	**0.00**	49.40	**0.00**	**0.00**	45.30	**0.00**	**0.00**	1425.88	**0.00**	**0.00**	537.00	**0.00**	473.59	**0.00**	0.29	533.61
C1_2_1	200	3628.51	0.42	41.00	0.04	0.20	87.38	**0.00**	0.22	2874.50	0.25	0.29	435.00	0.24	917.43	0.10	0.37	263.00
C2_2_1	200	1726.59	**0.00**	56.40	**0.00**	**0.00**	65.01	**0.00**	**0.00**	1365.93	**0.00**	**0.00**	331.20	**0.00**	398.38	**0.00**	0.05	437.95
RC1_2_1	200	3303.70	0.60	51.30	0.07	0.42	71.71	**0.00**	0.09	1293.53	0.02	0.35	486.00	0.07	485.05	**0.00**	0.19	267.61
RC2_2_1	200	1560.00	**0.00**	28.10	**0.00**	**0.00**	44.71	**0.00**	**0.00**	1361.87	**0.00**	**0.00**	441.00	**0.00**	393.51	**0.00**	0.13	532.29
Avg.			**0.15**	**26.88**	**0.02**	**0.10**	**37.47**	**0.00**	**0.03**	**799.66**	**0.03**	**0.10**	**267.45**	**0.04**	**264.29**	**0.03**	**0.23**	**224.63**
Processor type			T5500		Xeon			i7			Opteron 250			Xeon E5420		i5-6600		
Processor speed			1.66 GHz		2.66 GHz			2.93 GHz			2.40 GHz			2.50 GHz		3.30 GHz		
Passmark score			584		1015			1297			649			1073		2098		
t^c(s)			**7.48 × ?**		**18.13 × 256 × 50**			**494.36 × 10**			**82.73 × 10**			**135.17 × 6 × ?**		**224.63 × 10**		
R1_4_1	400	9519.45	1.81	345.30	1.05	1.34	481.61	**0.00**	0.21	9177.90	0.30	0.78	1471.80			0.38	1.09	2241.57
R2_4_1	400	3546.49	0.73	125.00	0.14	0.31	459.15	**0.00**	0.08	9086.79	**0.00**	0.24	1474.20			0.01	0.48	3066.73
C1_4_1	400	11047.19	1.20	224.80	0.47	0.65	546.22	**0.00**	0.26	8016.83	0.27	0.60	1780.20			0.33	0.64	2456.82
C2_4_1	400	3539.50	0.28	238.20	0.19	0.55	488.56	**0.00**	0.12	10691.30	**0.00**	0.05	1785.60			0.13	0.42	2600.01
RC1_4_1	400	9447.53	2.09	160.70	0.94	1.24	513.38	**0.00**	0.32	10867.10	0.23	0.65	1790.40			0.50	1.19	2542.94
RC2_4_1	400	3403.70	0.59	315.70	**0.00**	0.03	422.61	**0.00**	**0.00**	8326.18	**0.00**	0.01	1446.00			**0.00**	0.19	2166.65
Avg. tot.			**0.47**	**96.24**	**0.17**	**0.30**	**186.73**	**0.00**	**0.08**	**3653.44**	**0.07**	**0.20**	**719.87**			**0.09**	**0.37**	**987.24**
t^c(s)			**26.79 × ?**		**90.34 × 256 × 50**			**2258.59 × 10**			**222.69 × 10**					**987.24 × 10**		

Table A.3: Detailed results of ALNS-PR and the state-of-the-art heuristics for the VRPSPD on the Montané-All (including Montané-Medium) benchmark.

			SUO			PKKG			**ALNS-PR**			$\overline{\textbf{ALNS-PR}}$	
Inst.	n	BKS	Δ^b(%)	Δ^a(%)	t^a(s)	Δ^b(%)	Δ^a(%)	t(s)	Δ^b(%)	Δ^a(%)	t^a(s)	f	Δ(%)
CMT6X	50	555.43	**0.00**	0.35	1.04	**0.00**	**0.00**	47.00	**0.00**	**0.00**	7.70	**555.43**	0.00
CMT6Y	50	555.43	**0.00**	0.30	1.08	**0.00**	**0.00**	47.30	**0.00**	**0.00**	8.76	**555.43**	0.00
CMT7X	75	900.12	**0.00**	0.10	4.55	0.12	0.12	70.30	0.05	0.19	24.78	**900.12**	0.00
CMT7Y	75	900.12	**0.00**	0.11	4.87	0.12	0.12	69.80	**0.00**	0.08	21.86	**900.12**	0.00
CMT8X	100	865.50	**0.00**	**0.00**	7.36	**0.00**	**0.00**	224.60	**0.00**	**0.00**	49.24	**865.50**	0.00
CMT8Y	100	865.50	**0.00**	**0.00**	7.74	**0.00**	**0.00**	162.70	**0.00**	0.08	49.24	**865.50**	0.00
CMT14X	100	821.75	**0.00**	**0.00**	5.42	**0.00**	**0.00**	228.50	**0.00**	**0.00**	22.29	**821.75**	0.00
CMT14Y	100	821.75	**0.00**	**0.00**	5.48	**0.00**	**0.00**	204.60	**0.00**	**0.00**	25.64	**821.75**	0.00
CMT13X	120	1542.86	0.00	0.04	68.72	0.00	0.02	332.70	0.01	0.11	78.76	**1541.14**	-0.11
CMT13Y	120	1542.86	0.00	0.10	73.49	0.00	0.00	375.30	0.00	0.07	89.00	**1541.14**	-0.11
CMT9X	150	1160.68	**0.00**	0.09	64.43	0.05	0.09	483.40	**0.00**	0.23	116.35	**1160.68**	0.00
CMT9Y	150	1160.68	**0.00**	0.16	80.86	0.05	0.08	477.10	**0.00**	0.05	186.37	**1160.68**	0.00
CMT10X	199	1373.40	**0.00**	0.42	552.81	1.08	1.12	1168.80	**0.00**	1.04	324.94	**1373.40**	0.00
CMT10Y	199	1373.40	**0.00**	0.26	547.39	1.08	1.12	1112.10	**0.00**	1.11	243.13	**1373.40**	0.00
Avg.			**0.00**	**0.14**	**101.80**	**0.18**	**0.19**	**357.44**	**0.00**	**0.21**	**89.15**		**-0.02**
Processor type				i7			Core 2 Duo T5750			i5-6600			
Processor speed				2.93 GHz			2.00 GHz			3.30 GHz			
Passmark score				1297			678			2098			
t^c(s)				**62.93 × 10**			**115.51 × 10**			**89.15 × 10**			

Table A.4: Detailed results of ALNS-PR and the state-of-the-art heuristics for the VRPSPDTL on the Salhi-VRPSPDTL benchmark.

			PKKG			**ALNS-PR**			
Inst.	n	BKS	Δ^b(%)	Δ^a(%)	t^a(s)	f^b	Δ^b(%)	Δ^a(%)	t^a(s)
CE51-5	50	**570**	**0.00**	**0.00**	98.39	**570**	0.00	**0.00**	1.69
CE76-7	75	735	0.00	0.00	250.75	**728**	-0.95	0.11	6.17
CE76-8	75	778	0.00	0.35	507.78	**777**	-0.13	-0.10	4.81
CE76-10	75	**878**	**0.00**	0.17	344.06	**878**	0.00	0.18	4.10
CE76-14	75	**1091**	**0.00**	0.11	321.56	**1091**	0.00	0.10	4.46
CE101-8	100	956	0.00	0.05	416.56	**955**	-0.10	0.25	10.03
CE101-14	100	**1175**	**0.00**	0.30	10264	1177	0.17	0.56	13.71
Avg.			**0.00**	**0.14**	**323.18**		**-0.15**	**0.16**	**5.21**
Processor type			Core 2 Duo T5750			i5-6600			
Processor speed			2.00 GHz			3.30 GHz			
Passmark score			678			2098			
t^c(s)			**104.44** × **10**			**5.21** × **10**			

Table A.5: Detailed results of ALNS-PR in comparison to PKKG on the VRPSPDTL benchmark Polat-VRPSPDTL.

		BKS		WMZS				**ALNS-PR**			
Inst.	n	m	f	Δ^m	f^b	Δ^b(%)	t(s)	Δ^m	f^b	Δ^b(%)	t^a(s)
cdp101	100	11	992.88	0	992.88	0.00	36	0	**976.04**	-1.70	19.30
cdp102	100	10	955.31	0	955.31	0.00	38	0	**941.49**	-1.45	28.11
cdp103	100	10	897.65	0	958.66	6.80	34	0	**892.98**	-0.52	48.03
cdp104	100	10	878.93	0	944.73	7.49	35	0	**871.40**	-0.86	46.51
cdp105	100	11	983.10	0	989.86	0.69	37	**-1**	1053.12	7.12	15.97
cdp106	100	11	878.29	0	878.29	0.00	37	**-1**	967.71	10.18	17.36
cdp107	100	11	911.90	0	911.90	0.00	41	**-1**	987.64	8.31	18.06
cdp108	100	10	951.24	0	1063.73	11.83	39	0	**932.88**	-1.93	18.22
cdp109	100	10	940.49	0	947.90	0.79	21	0	**910.95**	-3.14	38.45
cdp201	100	3	591.56	0	591.56	0.00	86	0	**591.56**	0.00	24.37
cdp202	100	3	591.56	0	591.56	0.00	91	0	**591.56**	0.00	46.09
cdp203	100	3	591.17	0	591.17	0.00	88	0	**591.17**	0.00	44.19
cdp204	100	3	590.60	0	594.07	0.59	90	0	**590.60**	0.00	51.76
cdp205	100	3	588.88	0	588.88	0.00	90	0	**588.88**	0.00	36.35
cdp206	100	3	588.49	0	588.49	0.00	88	0	**588.49**	0.00	36.39
cdp207	100	3	588.29	0	588.29	0.00	85	0	**588.29**	0.00	39.83
cdp208	100	3	588.32	0	599.32	1.87	83	0	**588.32**	0.00	34.47
rdp101	100	19	1653.53	0	1660.98	0.45	43	0	**1650.80**	-0.17	22.86
rdp102	100	17	1488.04	0	1491.75	0.25	29	0	**1486.12**	-0.13	20.89
rdp103	100	14	1216.16	0	1226.77	0.87	41	**-1**	1297.01	6.65	17.83
rdp104	100	10	1000.65	0	1000.65	0.00	45	0	**984.81**	-1.58	28.40
rdp105	100	14	1399.81	0	1399.81	0.00	45	0	**1377.11**	-1.62	17.58
rdp106	100	12	1275.69	0	1275.69	0.00	37	0	**1252.03**	-1.85	27.54
rdp107	100	11	1082.92	0	1082.92	0.00	35	**-1**	1121.86	3.60	18.79
rdp108	100	10	962.48	0	962.48	0.00	41	**-1**	965.54	0.32	20.60
rdp109	100	12	1160.00	0	1181.92	1.89	46	**-1**	1194.73	2.99	16.08
rdp110	100	11	1106.52	0	1106.52	0.00	45	**-1**	1148.20	3.77	19.13
rdp111	100	11	1065.27	0	1073.62	0.78	41	**-1**	1098.84	3.15	22.15
rdp112	100	10	966.06	0	966.06	0.00	51	**-1**	1010.42	4.59	28.63
rdp201	100	4	1280.44	0	1286.55	0.48	84	0	**1253.23**	-2.12	33.44
rdp202	100	4	1100.92	0	1150.31	4.49	123	**-1**	1191.70	8.25	46.71
rdp203	100	3	950.79	0	997.84	4.95	102	0	**946.28**	-0.47	84.91
rdp204	100	2	848.01	0	848.01	0.00	120	0	**833.09**	-1.76	111.52
rdp205	100	3	1046.06	0	1046.06	0.00	116	0	**994.43**	-4.94	80.35
rdp206	100	3	959.94	0	959.94	0.00	134	0	**913.68**	-4.82	89.88
rdp207	100	2	899.82	0	899.82	0.00	85	0	**890.61**	-1.02	82.51
rdp208	100	2	739.06	0	739.06	0.00	127	0	**726.82**	-1.66	100.25
rdp209	100	3	930.26	0	947.80	1.89	111	0	**909.16**	-2.27	86.59
rdp210	100	3	983.75	0	1005.11	2.17	164	0	**939.37**	-4.51	82.84
rdp211	100	3	812.44	0	812.44	0.00	98	**-1**	904.44	11.32	85.87
rcdp101	100	15	1652.90	0	1659.59	0.40	47	**-1**	1776.58	7.48	10.59
rcdp102	100	13	1522.76	0	1522.76	0.00	41	**-1**	1583.62	4.00	19.19
rcdp103	100	11	1344.62	0	1344.62	0.00	45	0	**1283.52**	-4.54	29.08
rcdp104	100	10	1268.43	0	1268.43	0.00	47	0	**1171.65**	-7.63	22.57
rcdp105	100	14	1581.26	0	1581.54	0.02	46	0	**1548.96**	-2.04	16.95
rcdp106	100	13	1418.16	0	1418.16	0.00	41	**-1**	1392.47	-1.81	20.47
rcdp107	100	11	1360.17	0	1360.17	0.00	35	0	**1255.06**	-7.73	21.22
rcdp108	100	11	1169.57	0	1169.57	0.00	38	**-1**	1198.36	2.46	19.52
rcdp201	100	4	1513.72	0	1513.72	0.00	64	0	**1406.94**	-7.05	26.68
rcdp202	100	4	1211.12	0	1273.26	5.13	72	**-1**	1414.55	16.80	39.94
rcdp203	100	3	1123.58	0	1123.58	0.00	78	0	**1050.64**	-6.49	83.95
rcdp204	100	3	822.02	0	897.14	9.14	80	0	**798.46**	-2.87	94.85
rcdp205	100	4	1371.08	0	1371.08	0.00	62	0	**1297.65**	-5.36	33.58
rcdp206	100	3	1166.88	0	1166.88	0.00	66	0	**1146.32**	-1.76	59.36
rcdp207	100	3	1089.85	0	1089.85	0.00	75	0	**1061.84**	-2.57	79.06
rcdp208	100	3	862.89	0	862.89	0.00	73	0	**828.14**	-4.03	72.78
Avg.						**1.12**	**65.93**			**-2.32**	**42.12**
$\sum$				**0**				**-17**			
Processor type					2 x Xeon E5-2650				i5-6600		
Processor speed					2.00 GHz				3.30 GHz		
Passmark score					1910 (15283/8)				2098		
t^c(s)					**60.02** $\times$ **66** $\times$ **?**				**42.12** $\times$ **10**		

Table A.6: Detailed results of ALNS-PR in comparison to WMZS and the previous BKS on the VRPSPDTW benchmark Wang-Medium.

		WMZS			**ALNS-PR**				
Inst.	n	m	f^b	t(s)	m	Δ^m	f^b	Δ^b(%)	t^a(s)
C1_2_1	200	21	3169.52	62	**20**	-1	2846.20	-10.20	55.54
C1_4_1	400	42	8135.35	147	**40**	-2	7533.03	-7.40	220.13
C1_6_1	600	69	19720.65	257	**63**	-6	15594.21	-20.92	765.79
C1_8_1	800	88	32801.92	1054	**82**	-6	27035.71	-17.58	1495.17
C1_10_1	1000	110	52328.78	2418	**102**	-8	44764.64	-14.46	2253.40
C2_2_1	200	6	1972.97	112	6	0	**1931.44**	-2.10	194.73
C2_4_1	400	14	5085.08	279	**12**	-2	4144.84	-18.49	776.51
C2_6_1	600	20	9509.15	926	**18**	-2	7830.16	-17.66	1652.55
C2_8_1	800	27	14573.93	3636	**24**	-3	11759.05	-19.31	3022.47
C2_10_1	1000	33	23981.11	6529	**30**	-3	17088.50	-28.74	5009.45
R1_2_1	200	22	5083.39	89	**20**	-2	4849.80	-4.60	113.46
R1_4_1	400	42	12202.62	237	**40**	-2	10671.70	-12.55	537.22
R1_6_1	600	62	25729.28	581	**59**	-3	22306.17	-13.30	1211.12
R1_8_1	800	93	51949.49	1869	**80**	-13	39348.17	-24.26	3082.84
R1_10_1	1000	115	77993.35	4539	**100**	-15	58912.62	-24.46	4651.74
R2_2_1	200	5	4372.17	295	5	0	**4042.67**	-7.54	186.02
R2_4_1	400	9	14119.64	735	9	0	**8952.24**	-36.60	506.62
R2_6_1	600	13	27294.11	2439	13	0	**17459.41**	-36.03	1558.88
R2_8_1	800	19	48611.60	7663	**18**	-1	27270.04	-43.90	3429.95
R2_10_1	1000	22	67441.51	21379	22	0	**42117.48**	-37.55	5604.70
RC1_2_1	200	20	3865.18	81	**19**	-1	3652.18	-5.51	77.05
RC1_4_1	400	40	10036.82	193	**38**	-2	9772.56	-2.63	401.10
RC1_6_1	600	60	20535.26	733	**57**	-3	19679.75	-4.17	946.06
RC1_8_1	800	88	32801.92	1620	**75**	-13	38431.09	17.16	2234.79
RC1_10_1	1000	102	66883.49	3483	**93**	-9	63953.66	-4.38	4325.54
RC2_2_1	200	4	2662.75	216	4	0	**2021.49**	-24.08	338.41
RC2_4_1	400	13	7229.22	791	**12**	-1	6621.94	-8.40	681.44
RC2_6_1	600	20	22837.36	2860	**16**	-4	12693.19	-44.42	2607.91
RC2_8_1	800	22*	39375.78*	6026	60	38	26652.10	-32.31	2827.44
RC2_10_1	1000	29*	61473.68*	13793	75	46	40643.56	-33.88	6460.91
Avg.				**2834.73**				**-16.93**	**1907.63**
$\sum$						**-102**			
Processor type		2 × Xeon E5-2650			i5-6600				
Processor speed		2.00 GHz			3.30 GHz				
Passmark score		1910 (15283/8)			2098				
t^c(s)		**2580.71 × 66 × ?**			**1907.63 × 10**				

Table A.7: Detailed results of ALNS-PR in comparison to WMZS on the VRPSPDTW benchmark Wang-Large.

		Nagy1						**Nagy2**						**Nagy3**					
			ALNS-PR						**ALNS-PR**						**ALNS-PR**				
Inst.	n	BKS	m	n^d	f^b	$\Delta^b(\%)$	$t^a(\text{s})$	BKS	m	n^d	f^b	$\Delta^b(\%)$	$t^a(\text{s})$	BKS	m	n^d	f^b	$\Delta^b(\%)$	$t^a(\text{s})$
CMT1X	50	470	3	0	**468**	-0.43	9.16	1138	18	10	**1132**	-0.53	6.26	**1827**	32	25	**1827**	0.00	3.24
CMT1Y	50	**459**	3	0	**459**	0.00	8.35	1058	17	11	**1054**	-0.38	4.80	1841	32	31	**1838**	-0.16	3.85
CMT2X	75	684	6	1	**674**	-1.46	22.01	**1952**	34	17	1968	0.82	12.08	2742	48	38	**2738**	-0.15	8.48
CMT2Y	75	**650**	6	2	651	0.15	19.75	**1820**	30	22	1822	0.11	12.68	2809	47	53	**2799**	-0.36	10.11
CMT3X	100	713	5	1	**707**	-0.84	48.79	1946	28	26	**1938**	-0.41	31.78	3412	57	41	**3399**	-0.38	18.97
CMT3Y	100	705	4	0	**691**	-1.99	51.50	**1805**	27	25	1818	0.72	27.82	3425	58	49	**3415**	-0.29	18.33
CMT4X	150	862	7	1	**844**	-2.09	119.85	2742	43	55	**2714**	-1.02	102.58	4933	87	70	**4905**	-0.57	59.50
CMT4Y	150	831	7	0	**820**	-1.32	104.73	2548	40	36	**2525**	-0.90	76.57	4947	85	87	**4898**	-0.99	51.15
CMT5X	199	1062	10	16	**1015**	-4.43	203.04	3672	61	67	**3624**	-1.31	146.36	6449	119	109	**6388**	-0.95	100.30
CMT5Y	199	982	9	0	**966**	-1.63	182.71	3297	56	47	**3277**	-0.61	136.28	6568	116	129	**6430**	-2.10	115.73
CMT6X	50	**548**	6	0	**548**	0.00	5.01	1138	18	12	**1132**	-0.53	6.13	**1827**	32	25	**1827**	0.00	3.27
CMT6Y	50	**548**	6	0	**548**	0.00	5.78	1058	17	6	**1055**	-0.28	5.60	1841	31	32	**1832**	-0.49	4.15
CMT7X	75	897	11	0	**895**	-0.22	9.21	**1952**	34	17	1972	1.02	13.16	2742	48	40	**2738**	-0.15	8.41
CMT7Y	75	897	11	0	**894**	-0.33	9.66	**1820**	30	18	1836	0.88	14.73	2809	48	50	**2801**	-0.28	10.07
CMT8X	100	856	9	0	**848**	-0.93	26.91	1946	29	28	**1913**	-1.70	35.44	3412	58	43	**3400**	-0.35	22.35
CMT8Y	100	856	9	0	**844**	-1.40	25.92	**1805**	27	23	1815	0.55	27.70	3425	58	47	**3416**	-0.26	20.69
CMT9X	150	**1142**	14	0	1143	0.09	61.57	2742	44	53	**2713**	-1.06	78.91	4933	87	68	**4908**	-0.51	65.62
CMT9Y	150	**1143**	14	0	**1143**	0.00	57.22	2548	41	43	**2524**	-0.94	68.59	4947	86	86	**4907**	-0.81	53.92
CMT10X	199	1372	17	1	**1355**	-1.24	115.76	3672	61	82	**3609**	-1.72	162.93	6449	118	109	**6398**	-0.79	103.24
CMT10Y	199	1366	17	0	**1353**	-0.95	134.77	3297	56	49	**3288**	-0.27	131.46	6568	115	128	**6419**	-2.27	121.95
CMT11X	120	873	4	4	**819**	-6.19	118.57	3633	29	38	**3572**	-1.68	64.09	7427	68	62	**7321**	-1.43	33.43
CMT11Y	120	826	4	6	**764**	-7.51	108.97	**3178**	28	8	3196	0.57	22.28	7499	68	67	**7469**	-0.40	25.22
CMT12X	100	672	5	4	**646**	-3.87	54.44	2426	32	33	**2408**	-0.74	35.46	3950	60	52	**3925**	-0.63	16.59
CMT12Y	100	632	5	0	**627**	-0.79	41.12	2167	31	24	**2161**	-0.28	19.99	3897	59	63	**3896**	-0.03	16.76
CMT13X	120	1546	11	0	**1524**	-1.42	35.51	3633	29	48	**3569**	-1.76	58.70	7427	68	58	**7325**	-1.37	29.74
CMT13Y	120	1542	11	0	**1524**	-1.17	28.69	**3178**	28	10	3198	0.63	22.42	7499	69	66	**7456**	-0.57	25.40
CMT14X	100	**821**	10	0	**821**	0.00	17.13	2426	32	34	**2420**	-0.25	29.16	3950	60	54	**3925**	-0.63	16.34
CMT14Y	100	**819**	10	0	821	0.24	16.20	2167	31	23	**2161**	-0.28	19.06	3897	59	61	**3895**	-0.05	16.83
Avg.						**-1.42**	**58.65**					**-0.40**	**49.04**					**-0.61**	**35.13**
$\sum$			**234**	**36**					**951**	**865**					**1873**	**1743**			

Table A.8: Detailed results of ALNS-PR on the VRPDDP benchmarks Nagy1, Nagy2, and Nagy3.

			P		ALNS-PR			
Inst.	n	BKS	Δ^b(%)	t^{ab}(s)	n^d	f^b	Δ^b(%)	t^a(s)
SCA3-0	50	635.62	0.00	8.59	2	**629.84**	-0.91	27.31
SCA3-1	50	**697.84**	**0.00**	8.07	0	**697.84**	0.00	56.29
SCA3-2	50	**659.34**	**0.00**	12.33	0	**659.34**	0.00	40.64
SCA3-3	50	**680.04**	**0.00**	9.36	0	**680.04**	0.00	38.88
SCA3-4	50	**690.50**	**0.00**	7.14	0	**690.50**	0.00	27.37
SCA3-5	50	**659.90**	**0.00**	9.69	0	**659.90**	0.00	31.04
SCA3-6	50	**651.09**	**0.00**	7.21	0	**651.09**	0.00	32.71
SCA3-7	50	**659.17**	**0.00**	9.33	0	**659.17**	0.00	32.12
SCA3-8	50	**719.47**	**0.00**	8.93	0	**719.47**	0.00	26.19
SCA3-9	50	**681.00**	**0.00**	9.63	0	**681.00**	0.00	28.49
SCA8-0	50	**961.50**	**0.00**	9.38	0	**961.50**	0.00	18.18
SCA8-1	50	**1049.65**	**0.00**	9.44	0	**1049.65**	0.00	17.66
SCA8-2	50	**1039.64**	**0.00**	11.62	0	**1039.64**	0.00	19.82
SCA8-3	50	**979.13**	**0.00**	52.08	1	**979.13**	0.00	16.92
SCA8-4	50	**1065.49**	**0.00**	10.20	0	**1065.49**	0.00	18.67
SCA8-5	50	1027.08	0.00	13.02	1	**1022.02**	-0.49	15.31
SCA8-6	50	**969.50**	**0.00**	69.31	1	**969.50**	0.00	17.59
SCA8-7	50	1051.28	0.00	17.58	3	**1047.78**	-0.33	17.85
SCA8-8	50	**1071.18**	**0.00**	12.71	0	**1071.18**	0.00	17.31
SCA8-9	50	**1057.26**	**0.00**	100.79	1	**1057.26**	0.00	13.91
CON3-0	50	**616.52**	**0.00**	9.18	0	**616.52**	0.00	48.67
CON3-1	50	**554.47**	**0.00**	6.01	0	**554.47**	0.00	40.44
CON3-2	50	**518.00**	**0.00**	14.50	0	**518.00**	0.00	64.45
CON3-3	50	**591.19**	**0.00**	11.04	0	**591.19**	0.00	57.85
CON3-4	50	**588.79**	**0.00**	4.37	0	**588.79**	0.00	53.80
CON3-5	50	**563.70**	**0.00**	9.26	0	**563.70**	0.00	60.18
CON3-6	50	499.05	0.00	13.15	1	**498.55**	-0.10	62.10
CON3-7	50	**576.48**	**0.00**	8.43	0	**576.48**	0.00	47.41
CON3-8	50	523.05	0.00	6.61	1	**521.71**	-0.26	59.11
CON3-9	50	**578.25**	**0.00**	9.12	0	**578.25**	0.00	57.71
CON8-0	50	**857.12**	**0.00**	96.91	1	**857.12**	0.00	31.63
CON8-1	50	**739.44**	**0.00**	110.83	1	**739.44**	0.00	24.77
CON8-2	50	**706.51**	**0.00**	52.36	1	**706.51**	0.00	28.29
CON8-3	50	811.07	0.00	10.82	1	**808.66**	-0.30	39.34
CON8-4	50	**771.30**	**0.00**	40.25	1	**771.30**	0.00	26.86
CON8-5	50	**754.88**	**0.00**	7.81	0	**754.88**	0.00	31.99
CON8-6	50	**678.92**	**0.00**	6.10	0	**678.92**	0.00	29.12
CON8-7	50	**811.96**	**0.00**	11.40	0	**811.96**	0.00	31.33
CON8-8	50	**766.99**	**0.00**	31.09	1	**766.99**	0.00	29.68
CON8-9	50	797.69	0.00	102.94	6	**796.40**	-0.16	53.76
Avg.			**0.00**	**23.96**			**-0.06**	**34.82**
$\sum$					**23**			
Processor type			Xeon E5420			i5-6600		
Processor speed			2.50 GHz			3.30 GHz		
Passmark score			1073			2098		
t^c(s)			**12.25 × 6 × ?**			**34.82 × 10**		

Table A.9: Detailed results of ALNS-PR in comparsion to P on the VRPDDP benchmark Polat-VRPDDP1.

			P		ALNS-PR			
Inst.	n	BKS	Δ^b(%)	t^{ab}(s)	n^d	f^b	Δ^b(%)	t^a(s)
r101	100	**1009.95**	**0.00**	67.04	0	**1009.95**	0.00	122.14
r201	100	**666.20**	**0.00**	59.58	0	**666.20**	0.00	321.07
c101	100	**1220.18**	**0.00**	26.42	0	1220.99	0.07	132.86
c201	100	**662.07**	**0.00**	30.78	0	**662.07**	0.00	148.76
rc101	100	1058.94	0.00	33.39	2	**1057.40**	-0.15	126.81
rc201	100	**672.92**	**0.00**	32.62	0	**672.92**	0.00	211.76
R1_2_1	200	3353.80	0.04	1780.00	5	**3340.19**	-0.41	523.52
R2_2_1	200	**1665.58**	**0.00**	1401.84	0	**1665.58**	0.00	1248.30
C1_2_1	200	**3628.51**	0.04	4082.55	0	3632.31	0.10	410.28
C2_2_1	200	**1726.59**	**0.00**	1123.42	0	**1726.59**	0.00	1114.79
RC1_2_1	200	**3303.70**	0.04	2168.18	5	3304.81	0.03	457.95
RC2_2_1	200	**1560.00**	**0.00**	1074.29	0	**1560.00**	0.00	1635.74
Avg.			**0.01**	**990.01**			**-0.03**	**537.83**
$\sum$					**12**			
Processor type			Xeon E5420			i5-6600		
Processor speed			2.50 GHz			3.30 GHz		
Passmark score			1073			2098		
t^c(s)			**506.33** × **6** × **?**			**537.83** × **10**		

Table A.10: Detailed results of ALNS-PR in comparison to P on the VRPDDP benchmark Polat-VRPDDP2.