Large Multiple Neighborhood Search for the Soft-Clustered Vehicle-Routing Problem

Timo Hintsch[*,a]

[a]*Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*

Abstract

The soft-clustered vehicle-routing problem (SoftCluVRP) is a variant of the classical capacitated vehicle-routing problem. Customers are partitioned into clusters and all customers of the same cluster must be served by the same vehicle. In this paper, we present a large multiple neighborhood search for the SoftCluVRP. We design and analyze multiple cluster destroy and repair operators as well as two post-optimization components, which are both based on variable neighborhood descent. The first allows inter-route exchanges of complete clusters, while the second searches for intra-route improvements by combining classical neighborhoods (2-opt, Or-Opt, double-bridge) and the Balas-Simonetti neighborhood. Computational experiments show that our algorithm clearly outperforms the only existing heuristic approach from the literature. By solving benchmark instances, we provide 130 new best solutions for 220 medium-sized instances with up to 483 customers and prove 12 of them to be optimal.

Key words: Vehicle Routing, Clustered Vehicle Routing, Large neighborhood search

1. Introduction

The *soft-clustered vehicle-routing problem* (SoftCluVRP) is a variant of the well-known *capacitated vehicle-routing problem* (CVRP, Toth and Vigo, 2014) and has been introduced by Defryn and Sörensen (2017). It can be described as follows. The customers are grouped into disjoint clusters and all customers of a cluster must be served by the same vehicle (*soft-cluster constraints*). Visits to customers of the same cluster can be interrupted by visits to customers of other clusters. This is a relaxation of the *clustered vehicle-routing problem* (CluVRP, Sevaux and Sörensen, 2008) in which interruption is not allowed, but all customers of a cluster must be served contiguously (*hard-cluster constraints*). Hintsch and Irnich (2018a) have shown that this relaxation can decrease the costs of optimal solutions by 6.21 % on average for medium-sized instances, but finding optimal solutions is very difficult.

Both the SoftCluVRP and the CluVRP arise in practical scenarios, e.g., in parcel/small-package delivery in courier companies (Sevaux and Sörensen, 2008): Typically, customers are grouped into regional zones/districts (see Butsch *et al.*, 2014, for districting) and parcels are sorted into containers according to their corresponding district by ZIP codes. Note that the districting and thus the sorting policy are made on the tactical planning level and altered only occasionally. They are fixed before the actual demand distribution is known. Therefore, the clustering decision must be taken into account when the routing decision is made on the operational planning level. In the CluVRP each parcel from one container is delivered before delivering parcels from another container is allowed, while in the SoftCluVRP there are no such requirements.

The CluVRP is addressed by exact approaches (Pop *et al.*, 2012; Battarra *et al.*, 2014) and by several metaheuristics (Barthélemy *et al.*, 2010; Expósito Izquierdo *et al.*, 2013; Vidal *et al.*, 2015; Expósito-Izquierdo

*Corresponding author.
Email address: thintsch@uni-mainz.de (Timo Hintsch)

et al., 2016; Defryn and Sörensen, 2017; Hintsch and Irnich, 2018b; Pop *et al.*, 2018). To the best of our knowledge, only two approaches consider the SoftCluVRP. Hintsch and Irnich (2018a) presented an exact branch-and-price algorithm, which provides optimal solutions for instances with up to 420 customers and up to 52 clusters. Defryn and Sörensen (2017) suggested a two-level metaheuristics that originally was developed for the CluVRP. In this case, the low-level routing problem only considers the routing of customers inside a cluster (intra-cluster routing) and the high-level routing problem alters the position of clusters inside a route or moves clusters to another route (inter-cluster routing). This approach was adapted to the SoftCluVRP by allowing for customers to be moved to any position inside the current route at the lower level. Hence, the low-level routing considers intra-route moves of customers and the high-level routing considers inter-route moves of complete clusters. Both levels are solved by variable neighborhood search (VNS, Mladenović and Hansen, 1997).

The main contribution of the paper at hand is the design and computational analysis of a *large multiple neighborhood search* (LMNS, Pisinger and Ropke, 2007) for the SoftCluVRP. We will show that our new LMNS is able to improve the best known solutions for more than half of the considered medium-sized benchmark instances. In addition, we provide solutions for large-sized benchmarks that were not considered for the SoftCluVRP in the literature before.

Large neighborhood search (LNS, Shaw, 1998; Ropke and Pisinger, 2006b) has been shown to solve a wide range of routing problems successfully (see the survey by Pisinger and Ropke, 2010). Our approach combines the usage of multiple destroy and repair operators with two variable neighborhood descents (VNDs, Hansen and Mladenović, 2001) for post-optimization. The first VND allows for swapping and relocating complete clusters between routes, while the second VND improves single routes by classical neighborhoods (2-opt, Or-Opt, double-bridge) for the asymmetric traveling salesman problem (ATSP) as well as the Balas-Simonetti neighborhood (Balas, 1999; Balas and Simonetti, 2001). Although it is of exponential size, the Balas-Simonetti neighborhood can be searched in polynomial time.

The general design of our approach is adapted from the LMNS for the CluVRP presented by Hintsch and Irnich (2018b). However, important components of their approach are based on the exploitation of the hard-cluster constraints, for example the preprocessing of intra-cluster routes, a meta-representation with meta-nodes for the clusters, and a generalization of the Balas-Simonetti neighborhood. Since we consider soft-cluster constraints, major modifications are required for the destroy and repair operators as well as the post-optimization, resulting in a clearly differing algorithm (see Section 2).

We use the following notation: Let $V = \{0, \dots, n\}$ be the node set with the depot node 0 and the customer nodes $V \setminus \{0\} = \{1, \dots, n\}$ and let E be the edge set. Then, the SoftCluVRP can be defined on a complete undirected graph $G = (V, E)$. A fleet of m homogeneous vehicles with capacity Q is located at the depot 0. The nodes are partitioned into $N + 1$ clusters $V_0, V_1, V_2, \dots, V_N$, where $V_0 = \{0\}$ represents the *depot cluster* for convenience. A positive demand $d_h > 0$ is associated with every *customer cluster* indexed by $h \in H = \{1, 2, \dots, N\}$. The depot cluster V_0 has zero demand $d_0 = 0$. We define $n_h = |V_h|$ as the cardinality of cluster $h \in H \cup 0$. A non-negative routing cost c_{ij} is associated with each edge $\{i, j\} \in E$.

The task is to find m feasible routes visiting each customer exactly once and minimizing the total routing costs. According to the literature, each vehicle has to serve at least one cluster. Hence, a route r is feasible if

(i) it starts and ends at the depot node 0 and serves at least one cluster $h \in H$,
(ii) it visits each customer $i \in V_h$ exactly once if any customer $j \in V_h$ is visited by r, and
(iii) the demand of the visited clusters respects the vehicle capacity Q.

The remainder of this paper is structured as follows. In Section 2, the overall LMNS algorithm and all its components are described in detail. Comprehensive computational studies are summarized in Section 3. We analyze the effects of the destroy and repair operators as well as both post-optimization components. Moreover, we compare the results of the LMNS to the results generated by Defryn and Sörensen (2017). Final conclusions are drawn in Section 4.

2. LMNS for the SoftCluVRP

The general LNS procedure (for VRPs) works as follows: A feasible starting solution has to be given or created. Then, a *destroy operator* removes a subset of the customers from the current solution. Afterwards, these customers are reinserted by a *repair operator*, possibly at different positions or in different routes. The destroy and repair operators are applied repeatedly until a stopping criterion is met, while keeping track of the best solution found.

The number of customers to be removed can vary from iteration to iteration. In the basic version, it is increased if no improvement can be found for a specified number of iterations (Shaw, 1998), while Ropke and Pisinger (2006a) randomly choose the number of customers out of a given range in each iteration. After restoring the solution with the repair operator, it is accepted as the current solution based on an acceptance criterion. Shaw (1998) only accepts improving solutions, while Ropke and Pisinger (2006a,b) suggest to use a simulated annealing acceptance criterion.

As an extension, Ropke and Pisinger (2006a) introduced the *adaptive* LNS (ALNS) for the pickup and delivery problem with time windows. In each iteration, the destroy and the repair operator are selected randomly out of a set of multiple destroy and repair operators depending on a given weight per operator. The weights are updated according to the success of the respective operators in former iterations. LMNS (Pisinger and Ropke, 2007) also uses different destroy and repair operators, but in contrast to ALNS their given weights remain unchanged. Our approach is an adaptation of the LMNS for the CluVRP developed by Hintsch and Irnich (2018b). We adopt the record-to-record acceptance criterion and the idea of post-optimizing the repaired solution (which was first suggested by Ropke, 2009). However, due to the soft-cluster constraints, customers of clusters that are served by the same route can be visited in arbitrary order and a meta-representation of routes on a cluster level is not applicable. The meta-representation was an essential property of the LMNS by Hintsch and Irnich (2018b). Hence, we have to implement four major modifications:

(i) We cannot exploit the pre-computation of intra-cluster routes. Instead, we calculate feasible routes that include the depot and serve one cluster or a pair of clusters. These routes are used during the construction phase and possibly during the repair and the post-optimization phase.

(ii) The destroy and repair operators have to be tailored to the SoftCluVRP.

(iii) Similarly, new variants of the cluster neighborhoods are presented and combined in a VND for post-optimization (called `Clu-VND` in the following).

(iv) The generalized version of the Balas-Simonetti neighborhood, used during and after the VND, cannot be employed. Instead, we extend the post-optimization phase by a second VND which combines classical neighborhoods with the basic Balas-Simonetti neighborhood. This VND searches for intra-route improvements and is called `ATSP-VND` in the following.

In the following, we describe all our LMNS components. Section 2.1 presents improvement strategies for single routes, including the `ATSP-VND`, and Section 2.2 combines two neighborhoods that exchange clusters between different routes to another VND, called `Clu-VND`. In Section 2.3, we introduce our destroy and repair operators. Subsequently, the overall LMNS is summarized in Section 2.4.

2.1. ATSP Heuristics

In the SoftCluVRP, customers of a cluster $h \in H$ that are visited by the same route can be visited in an arbitrary order. Hence, the construction or improvement of a single route r can be considered as a traveling salesman problem (TSP, Gutin and Punnen, 2007), where the task is to find a cost-minimizing route, starting and ending in the depot, and visiting all customers in between. In the following, $\bar{n}$ denotes the number of customer nodes visited by a single route r.

In this section, we present a simple VND for the ATSP, which is used in our LMNS as a post-optimization component. It is based on three classical edge-exchange neighborhoods (2-opt, Or-opt, and double-bridge, see, e.g., Funke *et al.*, 2005) and the Balas-Simonetti neighborhood. Furthermore, we embed the VND in an iterated local search (ILS, Johnson *et al.*, 2007), which results in a combined ILS/VND similar to the algorithm presented by Irnich (2008). This procedure is used during the construction phase (see Section 2.4). Before explaining the `ATSP-VND` and the `Combined-ILS/VND`, we give a short description on the Balas-Simonetti neighborhood.

Balas-Simonetti neighborhood. The Balas-Simonetti neighborhood $\mathcal{N}_k^{BS}$ was introduced by Balas (1999) and is defined for a given integer parameter $k \geq 2$. Let $r = (r_0 = 0, r_1, \ldots, r_{\bar{n}}, r_{\bar{n}+1} = 0)$ be a feasible route. Then, if r_i precedes r_j in r by at least k positions, node r_i must also precede node r_j in a neighbor route $r' \in \mathcal{N}_k^{BS}(r)$. Hence, the Balas-Simonetti neighborhood $\mathcal{N}_k^{BS}(r)$ comprises all routes r' in which (i) $r'_0 = r_0$ and $r'_{\bar{n}+1} = r_{\bar{n}+1}$, and (ii) for all $i, j \in \{1, \ldots, \bar{n}\}$ with $i + k \leq j$, node r_i precedes node r_j also in r'. A layered auxiliary network is constructed to find the best neighbor route, where each *network node* represents a combination of a node r_i of the current route r and a (possibly new) position i' in route r', for which $i - k < i' < i + k$ holds. Each source-sink path in the auxiliary network represents a feasible neighbor route $r' \in \mathcal{N}_k^{BS}(r)$.

We use Neil Simonetti's code (written in `C` and available online at `http://www.andrew.cmu.edu/user/neils/`) to construct the auxiliary network (for details, we refer to Balas and Simonetti, 2001; Simonetti and Balas, 1996). Next, we briefly summarize the most important properties: The auxiliary network is independent of the current route r and needs to be constructed only once beforehand. Only the costs of the arcs in the auxiliary network have to be updated for a given input route. Although the neighborhood is of exponential size (for details see Gutin *et al.*, 2007, p. 233), the shortest source-sink path, representing the best neighbor route $r' \in \mathcal{N}_k^{BS}(r)$, can be found in $\mathcal{O}(\bar{n}k^2 2^k)$ time by dynamic programming. Thus, the computational effort is linear w.r.t. the route size $\bar{n}$. Moreover, if $k \geq \bar{n}$, the best neighbor represents the optimal solution of the ATSP route. However, the computational effort grows exponentially with k.

ATSP-VND. We combine the Balas-Simonetti neighborhood $\mathcal{N}_k^{BS}$ with three classical edge-exchange neighborhoods in a simple VND and search them in the order 2-opt, Or-opt, double-bridge, and Balas-Simonetti. All three classical edge-exchange neighborhoods can be searched in $\mathcal{O}(\bar{n}^2)$ time (see Glover, 1996). The result is a local optimum w.r.t. all four neighborhoods. Note that the SoftCluVRP is defined as a symmetric problem, but all four neighborhoods, and hence the VND, are applicable to the asymmetric case as well.

Combined-ILS/VND. Our `Combined-ILS/VND` uses the parameters n^{small} for the maximum number of customer nodes in a *small route*, It_{ILS} as the number of ILS iterations for improving larger routes, and k for the Balas-Simonetti neighborhood used during the VND. Depending on the number of customer nodes $\bar{n}$ the algorithm distinguishes three cases:

- $\bar{n} \leq 2$: There is nothing to do. The resulting route is a *pendulum tour* including the depot and the only customer node (or two customer nodes); note that we consider symmetric instances.
- $3 \leq \bar{n} \leq n^{small}$: We construct an arbitrary starting route r and search for the best neighbor route $r' \in \mathcal{N}_{\bar{n}}^{BS}(r)$ only once. Since we set $k = \bar{n}$ for the Balas-Simonetti neighborhood, the resulting route is already optimal.
- $\bar{n} > n^{small}$: The actual combination of ILS and VND similar to the procedure by Irnich (2008) is applied: First, a starting route is constructed by the nearest neighbor heuristic. Second, we iteratively call `ATSP-VND`(k) and permute the derived local optimum by two random double-bridge moves. The result of the permutation is used as the new starting solution for the next iteration. Overall, It_{ILS} iterations are executed, while keeping track of the best solution found.

2.2. Cluster Neighborhoods and VND

The goal of this section is to present a simple combination of two cluster neighborhoods, *Relocate* and *Swap*, within a VND. Both cluster neighborhoods are adapted from the CVRP, but always move complete clusters. They both remove and reinsert a single (*Relocate*) or two different (*Swap*) cluster(s).

To remove a cluster $h \in H$, all customers $i \in V_h$ have to be removed from their current route. After removing a customer, the preceding and succeeding customers are connected. Note that the route remains feasible if all customers $i \in V_h$ are removed.

The reinsertion of cluster h into a given route r is feasible if d_h, the demand of cluster h, does not exceed the residual capacity of r. Only feasible insertions are considered. To reinsert the cluster h, all customers $i \in V_h$ are sorted randomly. Then, they are inserted one after another by the Procedure `Best`

`Insert`. A single customer is inserted into the current route by minimizing the insertion cost. Note that the computational effort is bounded by $\mathcal{O}(n_{\max}n)$, where $n_{\max}$ is the size of the largest cluster.

Procedure `Best Insert`(V_h^{ran}, r)

Input: Randomly sorted customers V_h^{ran} of cluster $h \in H$,
a route r.
Output: Insertion costs and new route r including all customers in V_h

for $i \in V_h^{ran}$ **do**
  $c_{min} = \infty$
  $pos = -1$
  for $j = 0, \ldots, size(r) - 2$ **do**
    $cost = c_{r_j,i} + c_{i,r_{j+1}} - c_{r_j,r_{j+1}}$
    if $cost < c_{min}$ **then**
      $c_{min} = cost$
      $pos = j$
  $r = (r_0, \ldots, r_{pos}, i, r_{pos+1}, \ldots, r_{\bar{n}}, r_{\bar{n}+1})$

In the following we describe the *Relocate* and the *Swap* neighborhoods:

Relocate Neighborhood. The neighborhood $\mathcal{N}^{\text{reloc}}$ comprises all SoftCluVRP solutions that result from the removal of a cluster from its current route and the insertion of the same cluster into the same or another route by the Procedure `Best Insert`. The size of $\mathcal{N}^{\text{reloc}}$ is Nm, which is bounded by N^2 in the extreme case. Therefore, the complexity to search it is $\mathcal{O}(n_{\max}nN^2)$, when using `Best Insert`.

Swap Neighborhood. The neighborhood $\mathcal{N}^{\text{swap}}$ contains all SoftCluVRP solutions that result from the swapping of two clusters from two different routes. A swap of cluster g, currently visited by route r, and cluster h, currently visited by route s, is performed as follows: First, we remove all nodes $i \in V_g \cup V_h$ from their current route. Second, we perform `Best Insert`(V_g^{ran}, s) and `Best Insert`(V_h^{ran}, r). The size of $\mathcal{N}^{\text{swap}}$ grows quadratically with the number of clusters N and the computational effort is limited to $\mathcal{O}(2n_{\max}nN^2)$.

Both neighborhoods are combined within a VND, called `Clu-VND` in the following. As it is common practice, we start with the neighborhood that can be searched faster, the *Relocate* neighborhood $\mathcal{N}^{\text{reloc}}$. For both neighborhoods, we use a first improvement pivoting strategy.

2.3. LNS Operators

Here, we describe the different destroy (Section 2.3.1) and repair operators (Section 2.3.2) employed in our LMNS.

2.3.1. Destroy Operators

The destroy operators always remove complete clusters, which means that each customer $i \in V_h$ is removed if cluster h is removed. Removing a cluster is performed as described in the previous section. The percentage of clusters to be removed is defined by a parameter τ and we use four different destroy operators, similar to the operators applied by Hintsch and Irnich (2018b):

1. *Random destroy* removes τN clusters at random (Ropke and Pisinger, 2006a). (Note that τN is always rounded to the next integer. Here, we omit the corresponding formular for simplicity.)

2. *Related destroy* was introduced by Shaw (1998) and we adapt it to the presence of clusters: First, one cluster h is removed at random. Then, $\tau N - 1$ clusters closest to h are removed, too. The distance between two clusters V_g and V_h is defined as $\min_{(i,j) \in V_g \times V_h} c_{ij}$.

3. *Worst destroy* was introduced by Ropke and Pisinger (2006a) and is adapted for clusters: First, the improvement that would be realized if a cluster is removed from the current solution is calculated for each cluster $h \in H$ and sorted by decreasing improvement in a list L. Furthermore, we define the parameter $\rho^{worst} \geq 1$ to randomize the operator. Then, for τN iterations, we determine a uniformly distributed random number $y \in [0, 1)$, pick the cluster at position $\lfloor y^{\rho^{worst}} |L| \rfloor$ in L, and remove it from L and the current solution.

4. *Route destroy* removes one entire route at random. Note that the parameter τ is not used by this operator.

2.3.2. Repair Operators

Analogous to the destroy operators, the repair operators reinsert complete clusters. All operators use the same procedure to insert a given cluster h: If the destroy operator has reduced the number of routes and not every vehicle serves at least one cluster in the current solution, the given cluster is used to start a new route. Otherwise, for each route where h could be inserted w.r.t. the capacity, we evaluate the insertion costs for cluster h by the Procedure `Best Insert` as described in Section 2.2. Afterwards, the route with smallest insertion cost is chosen and cluster h is inserted as determined before. If cluster h cannot be inserted into any route because of the capacity constraint, the repair operator is stopped and the current solution remains infeasible. The operators only differ in the order the clusters are inserted:

1. *Random repair* reinserts all removed clusters in random order.

2. *Demand repair* reinserts all removed clusters in descending order of their demand.

3. *Randomized Demand repair* is a mixture of both other repair operators and all removed clusters are sorted according to their demand in descending order. Let L' be the list of sorted clusters. The following procedure is repeated until all clusters are reinserted: Similar to the *worst destroy* operator, we pick the cluster at position $\lfloor y^{\rho^{demand}} |L'| \rfloor$ from L', where the parameter $\rho^{demand} \geq 1$ is used to randomize the operator and $y \in [0, 1)$ is a uniformly distributed random number. The chosen cluster is reinserted to the current solution and removed from L'.

2.4. Overall LMNS Algorithm

Our overall LMNS approach combines all components described in Sections 2.1–2.3. Next, we describe the pseudo-code that is given in Algorithm 1:

In Step 1, we employ a *savings algorithm*, tailored to the SoftCluVRP, to construct a starting solution x. In contrast to the classical savings algorithm, a *pendulum tour* is defined as a route visiting all customers of one cluster, instead of visiting only one customer. For each cluster $h \in H$, we calculate a route, starting and ending in the depot, and visiting all customers $i \in V_h$ by applying the `Combined-ILS/VND` (Section 2.1) with the given input parameters. Moreover, the same is done for each pair of clusters $(g, h) \in H \times H$. Such a route visits all customers $i \in V_g \cup V_h$ of both clusters. The costs of the resulting routes are defined as $\hat{c}_h$ and $\hat{c}_{g,h}$, respectively, and savings values are calculated for each pair (g, h) as $sav_{g,h} = \hat{c}_g + \hat{c}_h - \hat{c}_{g,h}$.

Now, we construct routes as follows. As in the classical savings algorithm, the largest savings value $sav_{g,h}$ is chosen first. Instead of constructing real routes already at this stage, we only consider the corresponding clusters g and h to be part of the same route. A saving becomes infeasible if both clusters are already part of the same route or if the total demand of both routes exceeds the vehicle capacity Q. If the resulting number of routes exceeds the number of vehicles m, we compute a bin-packing solution based on the clusters (Valério de Carvalho, 1999). Finally, for each set of clusters that are considered to be part of the same route, either generated by the savings algorithm or by the bin-packing approach, we construct a route with the `Combined-ILS/VND`. Such a route visits all customers belonging to clusters that were assigned to that route.

Afterwards, the main loop (Steps 2–14) runs for It_{LMNS} iterations. Note that infeasible solutions can occur after the destroy/repair phase, but we only consider feasible solutions (Step 3). Furthermore, given a parameter ϵ_{post}, we only consider promising solutions that fulfill the record-to-record criterion $c(x) <$

Algorithm 1: LMNS algorithm for the SoftCluVRP

Input: Iterations It_{ILS} and It_{LMNS}
Parameters k_{sav} and k_{post} of Balas-Simonetti neighborhoods
Weights $(\psi^{random}, \psi^{related}, \psi^{worst}, \psi^{route})$ and $(\omega^{random}, \omega^{demand}, \omega^{ranDem})$ of destroy and repair operators
Parameters n^{small}, ϵ_{LMNS}, ϵ_{post}, $\tau_{\min}, \tau_{\max}$, ρ^{worst}, and ρ^{demand}

1 $x := x^{accepted} := x^{best}$:=Savings Algorithm(n^{small}, It_{ILS}, k_{sav})
2 **for** $iter := 1, \ldots, It_{\text{LMNS}}$ **do**
3   **if** x is feasible **then**
4   **if** AcceptanceCriterion1(ϵ_{post}, x, x^{best}) **then**
5             $x :=$ `Clu-VND`(x)
6             $x :=$ `ATSP-VND`(k_{post}, x)
7   **if** $c(x) < c(x^{best})$ **then**
8             $x^{best} := x$
9   **if** AcceptanceCriterion2(ϵ_{LMNS}, x, x^{best}) **then**
10             $x^{accepted} := x$
11   Randomly choose $\tau \in \{\tau_{\min}, \ldots, \tau_{\max}\}$
12   Randomly choose $\text{Op}^{destroy}$ according to weights $(\psi^{random}, \psi^{related}, \psi^{worst}, \psi^{route})$
13   Randomly choose Op^{repair} according to weights $(\omega^{random}, \omega^{demand}, \omega^{ranDem})$
14     $x := \text{Op}^{repair}(\rho^{demand}, \text{Op}^{destroy}(\tau, \rho^{worst}, x^{accepted}))$

$(1 + \epsilon_{\text{post}})\, c(x^{best})$ for post-optimization, see Step 4. The post-optimization is performed in Steps 5 and 6 with the `Clu-VND` from Section 2.2 followed by the `ATSP-VND` from Section 2.1. The latter is called for each route of the current solution x and with $k = k_{\text{post}}$.

In Steps 7–10, we possibly update the best solution found and/or the accepted solution depending on the second acceptance criterion. Again, we use a record-to-record acceptance criterion and the current solution is accepted if $c(x) < (1 + \epsilon_{\text{LMNS}})\, c(x^{best})$ is fulfilled for a given parameter ϵ_{LMNS}. Note that we always set $\epsilon_{\text{post}} \geq \epsilon_{\text{LMNS}}$. Steps 11–13 randomly choose the percentage τ of clusters to be removed as well as one destroy and one repair operator out of the seven operators presented in Section 2.3. Afterwards, the chosen operators are applied in Step 14, resulting in the new solution for the next iteration.

In both the `Clu-VND` as well as the repair operators, we deviate from the described procedure if cluster g is inserted into a route which currently serves no other or only one other cluster h. In such a case, our algorithm uses the route that was already derived by the `Combined-ILS/VND` during the savings algorithm, according to cluster g or the pair of clusters (g, h), respectively.

Furthermore, we enable our algorithm to stop prematurely if a time limit is given.

3. Computational Results

All computational results are obtained using a standard PC equipped with MS Windows 7, an Intel(R) Core(TM) i7-5930K CPU processor clocked at 3.5 GHz, and with 64 GB of main memory. Our algorithm is implemented in C++ and compiled in 64-bit single-thread code with MS Visual Studio 2015 in release mode. If necessary, CPLEX 12.8 is used to compute bin-packing solutions.

In Section 3.1, we introduce the considered benchmark instances and in Section 3.2, the parameters of our LMNS are tuned. Afterwards, the calibrated LMNS is analyzed and compared to the two-level VNS by Defryn and Sörensen (2017) on different instance sets (Sections 3.3 and 3.4). Results for large-sized instances that were not considered for the SoftCluVRP in the literature before are presented in Section 3.5.

3.1. SoftCluVRP Benchmark Instances

We test our LMNS algorithm on three benchmark sets that were used in previous studies in the literature. All SoftCluVRP benchmark sets were derived from CVRP benchmarks by defining θ as the desired average number of customers per customer cluster and building $N = \lceil (n+1)/\theta \rceil$ customer clusters (for details, see Fischetti *et al.*, 1997; Bektaş *et al.*, 2011). The first benchmark set was proposed by Bektaş *et al.* (2011). They adapted the CVRP benchmarks called `A`, `B`, `P`, and `GC` by choosing $\theta \in \{2, 3\}$, resulting in the two subsets `GVRP-2` and `GVRP-3`. Overall, 158 small- and medium-sized instances (available online at `http://www.personal.soton.ac.uk/tb12v07/gvrp.html`) with 16 to 262 nodes and 6 to 131 clusters were generated. The second benchmark set `Golden` is based on the well-known CVRP instances by Golden *et al.* (1998) and was generated by choosing $\theta = \{5, \ldots, 15\}$ for each of the 20 original instances `Golden1` to `Golden20`. It was provided by Battarra *et al.* (2014) and consists of 220 large-scale instances with 201 to 484 nodes and 14 to 97 clusters. The third set `Li` was generated by Vidal *et al.* (2015) using the CVRP instances of Li *et al.* (2005) and $\theta = 5$. It comprises 12 large-scale instances with 561 to 1201 nodes and 113 to 241 clusters. The number of vehicles m is given for each instance and it is not allowed to use less vehicles.

For each instance, our LMNS is run with ten different random seeds. The computation time is measured as the average over the ten runs. In the following, all computation times T are given in seconds. Furthermore, we define the *gap* in percentage between the solution value z and the best known solution BKS as $gap = 100(z - \text{BKS})/\text{BKS}$. The smallest gap found in the ten runs is given by *Gap Best*, while *Gap Avg.* refers to the average gap over the ten runs.

3.2. Parameter studies

In this section, we determine reasonable parameter settings for our LMNS algorithm to obtain high-quality solutions in fast computation times. As suggested by Ropke and Pisinger (2006a), we start with a setting found during pretests and then analyze the different components. First, we configure the SoftCluVRP tailored savings algorithm in Section 3.2.1. Afterwards, we determine the basic LMNS parameters, including the weights for the destroy and repair operators, and assess the usefulness of our post-optimization components in Sections 3.2.2 and 3.2.3. If not stated otherwise, we refer to a setting of our algorithm as $\text{LMNS}^{k_{\text{post}}}_{It_{\text{LMNS}}}$ or, if a time limit is given, as $\text{LMNS}^{k_{\text{post}}}_{It_{\text{LMNS}}}(maxTime)$, where $maxTime$ is the time limit in seconds.

3.2.1. Parameters for the Savings Algorithm

The only parameters that need to be set for the savings algorithm are those of the `Combined-ILS/VND`: $n^{small}, It_{\text{ILS}}, k_{\text{sav}}$. We simply adopt the parameter settings chosen by Hintsch and Irnich (2018b), which turned out to be a good tradeoff between solution quality and computational effort. In their approach, the `Combined-ILS/VND` was used to compute the shortest Hamiltonian path for each pair of nodes inside a cluster, which played an important role for the overall algorithm. In the paper at hand, it is only used during the construction phase. Although the derived routes might be used during the overall algorithm (see Section 2.4), the results are not crucial for our LMNS. In contrast to the approach by Hintsch and Irnich (2018b), they can be corrected by later steps.

Therefore, we do not invest much effort in adjusting these parameters and set $(n^{small}, It_{\text{ILS}}, k_{\text{sav}}) = (8, 50, 3)$. Hence, routes with up to $n^{small} = 8$ customer nodes are solved exactly by applying the Balas-Simonetti neighborhood only once. Otherwise, the ILS runs for $It_{\text{ILS}} = 50$ iterations using $k_{\text{sav}} = 3$ for the Balas-Simonetti neighborhood. This decision is supported by experiments conducted a posteriori: For the chosen parameters, the setup $\text{LMNS}^{3}_{10\,000}$, e.g., generates an average *Gap Best* of 0.029 % (*Gap Avg.* = 0.136 %) over all `GVRP` and `Golden` instances, while the geometric mean of the computation times is *Geo.* $T =$ 17.0. Increasing the number of iterations It_{ILS} from 50 to 100 even leads to an inferior solution quality with *Gap Best* = 0.039 % and *Gap Avg.* = 0.138 % with the same computational effort (*Geo.* $T =$ 17.0).

3.2.2. Parameters for the basic LMNS

In this section, we analyze the basic parameters of our LMNS, focusing on the destroy and repair operators. Pretests have shown that $(\epsilon_{\text{post}}, \epsilon_{\text{LMNS}}, \tau_{min}, \tau_{max}, \rho^{worst}, \rho^{demand}) = (0.1, 0.005, 10, 40, 3, 50)$ represent a good basic setting. It means that the current solution x is post-optimized by the `Clu-VND` and the `ATSP-VND` only if $c(x) \leq 1.1c(x^{best})$ and accepted as new current solution only if $c(x) \leq 1.005c(x^{best})$. The destroy operator removes between $\tau_{min} = 10\,\%$ and $\tau_{max} = 40\,\%$ of the clusters from the current solution, and $\rho^{worst} = 3$ and $\rho^{demand} = 50$ are chosen as the randomization values for the *Worst destroy* and the *Randomized Demand repair* operators, respectively.

To configure the weights $(\psi^{random}, \psi^{related}, \psi^{worst}, \psi^{route}, \omega^{random}, \omega^{demand}, \omega^{ranDem})$ of all destroy and repair operators, we set $k_{\text{post}} = 3$ (see Section 3.2.3 for analyses on k_{post}) and run our LMNS with 10 000 iterations and for several different setups. To limit the computational effort, we only consider the 158 `GVRP` instances for this series of experiments. The most important finding concerning the destroy operators is that the operator *Route destroy* is clearly inferior compared to all three other destroy operators. For example, if only one destroy operator is used (together with equally weighted repair operators), the average *Gap Best* is 0.939 % for *Route destroy*. Using *Random (Related, Worst) destroy* instead, the average *Gap Best* is reduced to 0.013 % (0.032 %, 0.020 %). On the basis of these results and further pretests we decide to use the *Route destroy* less often than the other three operators. Comparing only these three operators, they perform comparable. Hence, we choose $(\psi^{random}, \psi^{related}, \psi^{worst}, \psi^{route}) = (0.3, 0.3, 0.3, 0.1)$. Similarly, for the repair operators, we find that choosing equal weights turns out to be a good setup. The resulting average *Gap Best* is smaller than 0.001 % and the best out of ten runs finds the BKS for all but one instance. By using only one repair operator (together with the weights previously chosen for the destroy operators), *Gap Best* ranges from 0.004 % to 0.009 %.

Subsequently, we systematically test for the usefulness of each and every operator. We compare the chosen setup (called *All Operators*) to setups where one of the operators is disabled, but the ratio of the remaining operators is kept fixed. For example, if the *Worst destroy* is disabled, the weights for the destroy operators change to $(\psi^{random}, \psi^{related}, \psi^{route}) = (0.3, 0.3, 0.1)/0.7$. Again, we set $k_{\text{post}} = 3$ and $It_{\text{LMNS}} = 10\,000$. The results are summarized in Table 1. *Avg. T* refers to the arithmetic mean of the average computation time over ten runs for all 158 instances and *Geo. T* gives the geometric mean.

	w/o destroy operator				w/o repair operator			*All*
	Random	*Related*	*Worst*	*Route*	*Random*	*Demand*	*Ran.Dem.*	*Operators*
Avg. T	3.1	3.1	3.1	3.0	3.1	3.1	3.0	3.2
Geo. T	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.1
Gap Best [%]	0.004	0.014	0.004	0.014	0.014	0.026	0.003	<0.001
Gap Avg. [%]	0.093	0.111	0.100	0.081	0.123	0.090	0.092	0.086
# BKS (158)	156	154	156	155	154	155	156	157

Table 1: Comparison of LMNS using different destroy and repair operators and 158 `GVRP` benchmark instances.

The computational effort is nearly the same for all settings. For example, *Avg. T* ranges from 3.0 to 3.2 seconds. *All Operators* produces an average *Gap Best* smaller than 0.001 %, while the other seven settings result in a *Gap Best* between 0.003 % and 0.026 %. Comparing for the average *Gap Avg.*, *All Operators* is inferior to the setting without *Route destroy* (0.086 % vs. 0.081 %), while the remaining six gaps are not smaller than 0.090 %. Nevertheless, due to the smaller *Gap Best*, we still keep the *Route destroy* operator. Furthermore, it is the only setting that finds the BKS in 157 out of the 158 `GVRP` instances. Hence, we fix the chosen weights for all remaining studies.

3.2.3. Usefulness of the post-optimization

In our LMNS, the post-optimization of repaired solutions comprises two components: The `Clu-VND`, which relocates and swaps complete clusters, and the `ATSP-VND`, which improves single routes and consists of four neighborhoods (2-opt, Or-opt, double-bridge, Balas-Simonetti). In addition to the `GVRP` instances, we

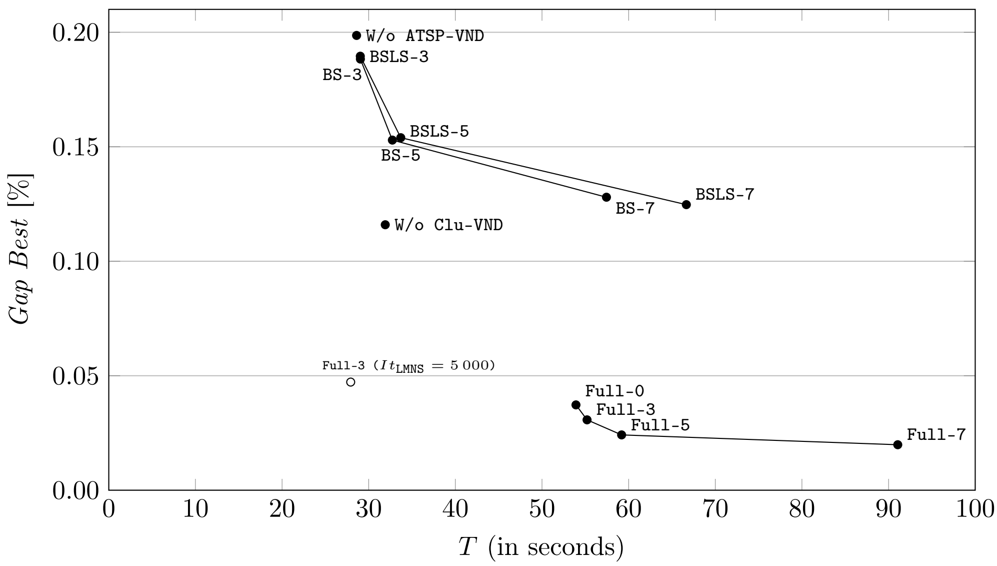

Figure 1: Comparison of different post-optimization strategies on 378 benchmarks (158 `GVRP` and 220 `Golden` instances).

include the medium-sized `Golden` instances for the analysis of the two components. Altogether, we compare the following 15 post-optimization strategies:

- `Full`-k_{post}: The full LMNS is applied as described in Section 2.4, including the `Clu-VND` and the `ATSP-VND`. For the `ATSP-VND`, we test 5 different settings with $k_{\text{post}} \in \{0, 3, 5, 7, 9\}$, where $k_{\text{post}} = 0$ means that the Balas-Simonetti neighborhood is switched off. Note that $k_{\text{post}} = 1$ is not reasonable because nodes could only be moved for *less* than $k_{\text{post}} = 1$ positions in the current route, which corresponds to not move them at all.
- `BSLS`-k_{post}: Instead of using the complete `ATSP-VND`, we only search for the local optimum with respect to the Balas-Simonetti neighborhood for each route, using different $k_{\text{post}} \in \{3, 5, 7, 9\}$. Note that $k_{\text{post}} = 0$ would switch off the `ATSP-VND` completely, which is the strategy `W/o ATSP`.
- `BS`-k_{post}: As `BSLS`-k_{post}, but instead of searching for the local optimum, the Balas-Simonetti neighborhood is applied only once for each route.
- `W/o ATSP`: The complete `ATSP-VND` is switched off. Only `Clu-VND` is used for post-optimization.
- `W/o Clu-VND`: `Clu-VND` is switched off. Only the (full) `ATSP-VND` with $k_{\text{post}} = 3$ is used for post-optimization.

We have also tested settings where the `Clu-VND` and `ATSP-VND` are incorporated within one VND or where the complete `Combined-ILS/VND` is performed, instead of only running the `ATSP-VND`. These were clearly inferior and we do not include them in the analysis of this section.

Figure 1 gives a comparison of the different post-optimization strategies when running the LMNS with 10 000 iterations. It reports the average computation time *Avg. T* and the average *Gap Best* for all the 378 `GVRP` and `Golden` instances. To compare for similar computation times, we additionally run the setting `Full-3` with a reduced number of iterations $It_{\text{LMNS}} = 5\,000$, indicated by the open dot $\circ$.

The results can be summarized as follows: First, it is superior to post-optimize solutions with the `ATSP-VND` including the three classical edge-exchange neighborhoods. All settings where `ATSP-VND` is switched off or reduced to a local/single search with the Balas-Simonetti neighborhood are clearly outperformed w.r.t. *Gap Best.* Reducing the number of iterations, e.g., of setting `Full-3`, shows that this also holds when computation times are comparable. Second, a similar effect is observed for the `Clu-VND`. Comparing the strategies `W/o Clu-VND` and `Full-3`, the `Clu-VND` decreases the *Gap Best* from 0.116 % to 0.031 % but increases the

Avg. T from 31.9 to 55.2 seconds. As before, reducing the number of iterations for `Full-3` helps to show the usefulness of `Clu-VND` by producing a *Gap Best* of 0.047 % in 27.9 seconds average computation time. Third, we observe the expected tradeoff between solution quality and computation time for the k_{post} of the Balas-Simonetti neighborhood used in the `ATSP-VND`. Higher values for k_{post} help to find better solutions, while the computational effort only raises reasonably for $k_{\text{post}} \leq 5$. However, for $k_{\text{post}} > 5$, computation times increase drastically with only little improvement in the solution quality. We omit settings with $k_{\text{post}} = 9$ in Figure 1 due to very high running times. For example, `Full-9` gives the smallest *Gap Best* of only 0.018 % but runs for 276.3 seconds on average.

Very similar results are observed by comparing *Gap Avg.* instead of *Gap Best*. Overall, both VND components used for post-optimization contribute to the quality of our LMNS. Furthermore, setting k_{post} to 3 or 5 yields the most favorable results and we report more details on both settings in the next sections. This result is very similar to observations from the literature (see, e.g., Gschwind and Drexl, 2018; Hintsch and Irnich, 2018b).

3.3. Results for the `GVRP` Instances

In this section, we give more detailed results of our LMNS for the small-sized `GVRP` instance set and compare them to the results of the two-level VNS proposed by Defryn and Sörensen (2017). Our LMNS is run with both chosen settings, $k_{\text{post}} = 3$ as well as $k_{\text{post}} = 5$, and again for $It_{\text{LMNS}} = 10\,000$ iterations.

	$\text{LMNS}^3_{10\,000}$					$\text{LMNS}^5_{10\,000}$					DS (2017)			
	T		*Gap*		#	*T*		*Gap*		#	*T*	*Gap*		#
Set (# inst.)	*Avg.*	*Geo.*	*Best*	*Avg.*	BKS	*Avg.*	*Geo.*	*Best*	*Avg.*	BKS	*Avg.*	*Best*	*Avg.*	BKS
`GVRP-2`														
`A-2 (27)`	1.87	1.71	0.00	0.11	27	2.78	2.54	0.00	0.10	27	n.a.	n.a.	n.a.	n.a.
`B-2 (23)`	2.19	2.05	0.00	0.01	23	2.91	2.76	0.00	0.01	23	n.a.	n.a.	n.a.	n.a.
`P-2 (24)`	2.42	2.42	0.00	0.14	24	3.18	2.06	0.00	0.12	24	n.a.	n.a.	n.a.	n.a.
`GC-2 (5)`	12.74	11.79	0.01	0.60	4	14.54	13.77	0.16	0.61	3	n.a.	n.a.	n.a.	n.a.
`GVRP-3`														
`A-3 (27)`	1.97	1.84	0.00	0.03	27	2.79	2.64	0.00	0.03	27	0.28	0.07	0.14	26
`B-3 (23)`	2.25	2.25	0.00	0.02	23	3.08	2.92	0.00	0.02	23	0.06	0.00	0.00	23
`P-3 (24)`	2.48	1.64	0.00	0.02	24	3.28	2.31	0.00	0.02	24	0.52	0.10	0.16	19
`GC-3 (5)`	22.71	17.98	0.00	0.49	5	24.95	20.22	0.00	0.44	5	13.46	0.43	0.91	1
Total (158)	3.17	2.06	<0.01	0.09	157	4.06	2.84	<0.01	0.08	156	n.a.	n.a.	n.a.	n.a.

Table 2: Aggregated results for the 158 `GVRP` instances.

The results are summarized in Table 2, where 'DS (2017)' refers to the two-level VNS by Defryn and Sörensen (2017). Our two LMNS settings perform very similar on these instances. In total, the computation time is smaller for $\text{LMNS}^3_{10\,000}$, but it differs less than one second on average. Overall, $\text{LMNS}^3_{10\,000}$ ($\text{LMNS}^5_{10\,000}$) finds the BKS for all but one (two) instance(s), resulting in an average *Gap Best* of <0.01 % (<0.01 %). For the `GC-2` instances, we obtain an average *Gap Best* of 0.01 % (0.16 %). Considering *Gap Avg.*, $\text{LMNS}^5_{10\,000}$ performs slightly better (0.08 % vs. 0.09 % overall).

Comparing with the two-level VNS, note that Defryn and Sörensen (2017) run their algorithm for 20 different random seeds, but did not consider the `GVRP-2` instances. Furthermore, they reported computation times only by arithmetic means for subsets. For the `GVRP-3` instances, they can find the BKS for 69 instances, whereas both LMNS settings find each and every BKS, resulting in smaller or equal *Gap Best* values. The *Gap Avg.* values are smaller for the LMNS, too, except for the `B-3` instances (0.02 % vs. 0.00 %). Computation times are clearly smaller for the two-level VNS, but also reasonably small for both LMNS settings and all subsets `A-3`, `B-3`, `P-3` (at most 3.28 seconds on average). Moreover, reducing the number of iterations to 1 000 and applying additional tests, for example with $\text{LMNS}^3_{1\,000}$, leads to average computation times that are smaller for our LMNS for each of the subsets except `B-3` (0.27 vs. 0.06 seconds), and we can still find every BKS.

Furthermore, our LMNS finds every solution that is known to be optimal (for optimal solutions, see Hintsch and Irnich, 2018a) and improves the BKS from the literature for 10 ($\text{LMNS}^3_{10\,000}$) and 11 ($\text{LMNS}^5_{10\,000}$) of the remaining 13 instances (where the exact approach was prematurely terminated after 3 600 seconds). One of the new BKS can even be proven to be the optimal solution, since the calculated costs equal the corresponding lower bound reported for this instance by Hintsch and Irnich (2018a). Detailed instance-by-instance results are given in Tables 5–8 of the Online Supplement.

3.4. Results for the `Golden` Instances

Analogous to the previous section, we analyze our LMNS for the medium-sized `Golden` instances. Results are given in Tables 3 (for $k_{\text{post}} = 3$) and 4 ($k_{\text{post}} = 5$), where the instances are grouped by average cluster size $\theta \in \{5, \ldots, 15\}$. Instances are easier to solve for larger average cluster sizes θ, which implies a decreasing number of clusters N. This leads to strictly decreasing computation times for both LMNS settings. The gaps also tend to decrease with an increasing θ, but this observation is ambiguous, in particular for *Gap Best*.

Over all 220 instances, the average *Gap Best* of 0.05 % produced by $\text{LMNS}^3_{10\,000}$ can be reduced to 0.04 % by $\text{LMNS}^5_{10\,000}$, accepting a slightly higher computation time (92.6 vs. 98.8 seconds on average and 77.6 vs. 84.6 geometrical mean). Simultaneously, *Gap Avg.* reduces from 0.18 % to 0.15 %. We observe smaller gaps for $\text{LMNS}^5_{10\,000}$ compared to $\text{LMNS}^3_{10\,000}$ for all average cluster sizes, except $\theta = 5$, where $\text{LMNS}^5_{10\,000}$ generates *Gap Best* = 0.05 % compared to 0.04 %. Overall, 147 (162) BKS are found by $\text{LMNS}^3_{10\,000}$ ($\text{LMNS}^5_{10\,000}$).

	$\text{LMNS}^3_{10\,000}$					$\text{LMNS}^3_{10\,000}(10)$				DS (2017)			
	Gap		*T*		#	*Gap*		*T*	#	*Gap*		*T*	#
θ	*Best*	*Avg.*	*Avg.*	*Geo*	BKS	*Best*	*Avg.*	*Avg.*	BKS	*Best*	*Avg.*	*Avg.*	BKS
5	0.04	0.18	126.5	105.3	14	0.30	0.71	10.0	6	3.84	5.02	10.0	0
6	0.08	0.24	113.2	94.9	10	0.35	0.64	10.0	4	3.58	4.65	10.0	0
7	0.03	0.19	105.1	89.5	14	0.29	0.58	10.0	5	3.31	4.24	10.0	0
8	0.07	0.18	98.5	84.1	14	0.22	0.45	10.0	9	3.01	3.97	10.0	0
9	0.04	0.16	92.5	79.0	14	0.22	0.51	10.0	8	2.66	3.65	10.0	0
10	0.04	0.16	88.5	76.0	14	0.17	0.44	10.0	9	2.77	3.51	10.0	0
11	0.09	0.17	84.7	72.4	10	0.18	0.44	10.0	9	2.60	3.38	10.0	0
12	0.08	0.17	82.0	70.3	12	0.20	0.40	10.0	9	2.45	3.20	10.0	0
13	0.07	0.24	79.3	67.7	14	0.16	0.44	10.0	7	2.42	3.26	10.0	0
14	0.03	0.12	75.0	64.2	15	0.16	0.34	10.0	8	2.28	3.10	10.0	0
15	0.02	0.10	72.9	61.8	16	0.12	0.31	10.0	8	2.27	3.11	10.0	0
Total	0.05	0.18	92.6	77.6	147	0.22	0.48	10.0	82	2.84	3.74	10.0	0

Table 3: Aggregated results for $k_{\text{post}} = 3$ and benchmark set `Golden`, sorted by the average number of nodes per cluster θ (220 instances divided into 11 groups of 20 instances each).

Since Defryn and Sörensen (2017) set a time limit of 10 seconds, we also run our LMNS with the same time limit. The results clearly show the superiority of our LMNS over the two-level VNS. For all groups of instances, the gaps obtained by the LMNS do not exceed 0.35 % for *Gap Best* and 0.73 % for *Gap Avg.*. On the contrary, Defryn and Sörensen (2017) report gaps between 2.27 % and 3.84 % (*Gap Best*), and from 3.10 % to 5.02 % (*Gap Avg.*). Moreover, $\text{LMNS}^3_{10\,000}(10)$ and $\text{LMNS}^5_{10\,000}(10)$ find 82 and 89 BKS, respectively, while the two-level VNS cannot find any BKS. Comparing the two LMNS settings with the time limit of ten seconds, $k_{\text{post}} = 5$ also performs slightly better w.r.t. both *Gap Best* and *Gap Avg.* (0.20 % and 0.45 % over all 220 `Golden` instances compared to 0.22 % and 0.48 %). However, comparing for different average cluster sizes, there is more volatility than for the case without a time limit.

Finally, both our LMNS settings produce 130 new BKS for the `Golden` instance set. Out of them, 7 solutions can be proven to be optimal because they hit the corresponding lower bound generated by the branch-and-price algorithm of Hintsch and Irnich (2018a). In addition, 5 solutions that were generated

	$\text{LMNS}^5_{10\,000}$					$\text{LMNS}^5_{10\,000}(10)$				DS (2017)			
	Gap		*T*		#	*Gap*		*T*	#	*Gap*		*T*	#
θ	*Best*	*Avg.*	*Avg.*	*Geo*	BKS	*Best*	*Avg.*	*Avg.*	BKS	*Best*	*Avg.*	*Avg.*	BKS
5	0.05	0.19	130.7	111.4	13	0.33	0.73	10.0	5	3.84	5.02	10.0	0
6	0.02	0.19	119.7	101.9	15	0.31	0.64	10.0	4	3.58	4.65	10.0	0
7	0.03	0.16	112.2	97.3	15	0.25	0.50	10.0	6	3.31	4.24	10.0	0
8	0.07	0.16	104.5	90.7	15	0.23	0.41	10.0	8	3.01	3.97	10.0	0
9	0.01	0.15	98.5	85.5	19	0.16	0.47	10.0	11	2.66	3.65	10.0	0
10	0.04	0.13	95.3	83.2	14	0.16	0.38	10.0	9	2.77	3.51	10.0	0
11	0.05	0.15	91.0	79.2	14	0.21	0.40	10.0	9	2.60	3.38	10.0	0
12	0.07	0.15	88.7	77.4	12	0.22	0.37	10.0	10	2.45	3.20	10.0	0
13	0.05	0.20	85.7	74.8	14	0.12	0.41	10.0	10	2.42	3.26	10.0	0
14	0.03	0.10	81.1	70.6	13	0.13	0.30	10.0	9	2.28	3.10	10.0	0
15	0.01	0.09	79.1	68.7	18	0.11	0.29	10.0	8	2.27	3.11	10.0	0
Total	0.04	0.15	98.8	84.6	162	0.20	0.45	10.0	89	2.84	3.74	10.0	0

Table 4: Aggregated results for $k_{\text{post}} = 5$ and benchmark set `Golden`, sorted by the average number of nodes per cluster θ (220 instances divided into 11 groups of 20 instances each).

during the computational experiments and further improved the BKS are also proven to be optimal. Overall, our LMNS with setting $\text{LMNS}^3_{10\,000}$ ($\text{LMNS}^5_{10\,000}$) finds 81 (82) out of the 99 solutions for `Golden` instances that are now known to be optimal. Detailed results for each instance are given in the Online Supplement (Tables 9–12).

Further note that, compared to the BKS reported for the CluVRP in the literature, heuristic solutions generated with our LMNS for the SoftCluVRP (e.g. with setting $\text{LMNS}^5_{10\,000}$) reduce the costs by 6.19 % on average over all `Golden` instances. If we only consider instances that are solved exactly for both problem variants, the cost reduction is 6.10 % on average. We refer to Hintsch and Irnich (2018a) for a more detailed comparison of hard- and soft-cluster constraints on exactly solved instances.

3.5. Results for the `Li` instances

In a subsequent study, we run our LMNS with both settings, $\text{LMNS}^3_{10\,000}$ and $\text{LMNS}^5_{10\,000}$, on the 12 large-sized `Li` instances (see Table 13 of the Online Supplement for detailed instance-by-instance results). These were not solved for the SoftCluVRP before. $\text{LMNS}^3_{10\,000}$ finds the better result for 7 instances, while $\text{LMNS}^5_{10\,000}$ finds better solutions for the remaining 5 instances. The resulting gaps are *Gap Best* = 0.02 % (*Gap Avg.* = 0.36 %) within 658 seconds of average runtime for $\text{LMNS}^3_{10\,000}$ and *Gap Best* = 0.03 % (*Gap Avg.* = 0.31 %) within 680 seconds for $\text{LMNS}^5_{10\,000}$. Hence, $\text{LMNS}^3_{10\,000}$ performs slightly better on these instances, but note that they were all generated by choosing $\theta = 5$ and $\text{LMNS}^3_{10\,000}$ also performed better on this group of the `Golden` instances.

Compared to the BKS for the CluVRP (see Vidal *et al.*, 2015; Hintsch and Irnich, 2018b), costs for the `Li` instances are reduced by up to 7.38 % (4.75 % on average) due to the relaxation of only including soft-cluster constraints.

4. Conclusions

In this article, we designed and analyzed a new and well-structured LMNS for the SoftCluVRP. For our new LMNS we presented four destroy and three repair operators, all tailored to the SoftCluVRP. These are used to remove and reinsert complete clusters during the destroy and repair phase. Furthermore, we added two post-optimization components to improve restored solutions after the repair step by local search. Both components are based on VND. The first VND uses new variants of cluster neighborhoods that allow the exchange of clusters between routes, while the second VND improves single routes with the help of classical edge-exchange neighborhoods and the Balas-Simonetti neighborhood.

We have carefully tested our algorithm on benchmark instances from the literature, showing that all components, in particular both the Clu-VND and the ATSP-VND, help to increase the quality of our LMNS. Our algorithm clearly outperforms the two-level VNS by Defryn and Sörensen (2017), the only existing metaheuristic from the literature. For the medium-sized Golden instances, e.g., our algorithm produces an average gap of 0.45 % (best gap of 0.20 %) compared to 3.74 % (2.84 %) within the same time limit of ten seconds. Moreover, for more than half of these instances we generated new best known solutions. In addition, we could prove 13 new best solutions for small- and medium-sized benchmark instances to be optimal and our LMNS found 228 of 255 solutions that are known to be optimal. Furthermore, we provided solutions for large-sized instances with up to 1 200 customers and 241 clusters. These were not considered for the SoftCluVRP by the literature before, but comparing to best known solutions for the CluVRP (with hard-cluster constraints), costs were reduced by 4.75 % on average if only soft-cluster constraints have to be respected.

References

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, **86**(0), 529–558.

Balas, E. and Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing*, **13**(1), 56–75.

Barthélemy, T., Rossi, A., Sevaux, M., and Sörensen, K. (2010). Metaheuristic approach for the clustered VRP. In *EU/ME 2010 – 10th anniversary of the metaheuristic community*, Lorient, France.

Battarra, M., Erdoğan, G., and Vigo, D. (2014). Exact algorithms for the clustered vehicle routing problem. *Operations Research*, **62**(1), 58–71.

Bektaş, T., Erdoğan, G., and Ropke, S. (2011). Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. *Transportation Science*, **45**(3), 299–316.

Butsch, A., Kalcsics, J., and Laporte, G. (2014). Districting for arc routing. *INFORMS Journal on Computing*, **26**(4), 809–824.

Defryn, C. and Sörensen, K. (2017). A fast two-level variable neighborhood search for the clustered vehicle routing problem. *Computers & Operations Research*, **83**, 78–94.

Expósito Izquierdo, C., Rossi, A., and Sevaux, M. (2013). Modeling and Solving the Clustered Capacitated Vehicle Routing Problem. In A. Fink and M.-J. Geiger, editors, *Proceedings of the 14th EU/ME workshop, EU/ME 2013*, pages 110–115, Hamburg, Germany.

Expósito-Izquierdo, C., Rossi, A., and Sevaux, M. (2016). A two-level solution approach to solve the clustered capacitated vehicle routing problem. *Computers & Industrial Engineering*, **91**, 274–289.

Fischetti, M., González, J. J. S., and Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, **45**(3), 378–394.

Funke, B., Grünert, T., and Irnich, S. (2005). Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics*, **11**(4), 267–306.

Glover, F. (1996). Finding a best traveling salesman 4-opt move in the same time as a best 2-opt move. *Journal of Heuristics*, **2**(2), 169–179.

Golden, B. L., Wasil, E. A., Kelly, J. P., and Chao, I.-M. (1998). The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results. In T. G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 33–56. Springer US, Boston, MA.

Gschwind, T. and Drexl, M. (2018). Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, **forthcoming**.

Gutin, G. and Punnen, A. P., editors (2007). *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimization*. Springer US, Boston, MA.

Gutin, G., Yeo, A., and Zverovich, A. (2007). Exponential neighborhoods and domination analysis for the TSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimization*, pages 223–256. Springer US, Boston, MA.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, **130**(3), 449–467.

Hintsch, T. and Irnich, S. (2018a). Exact solution of the soft-clustered vehicle-routing problem. Technical Report LM-2018-04, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

Hintsch, T. and Irnich, S. (2018b). Large multiple neighborhood search for the clustered vehicle-routing problem. *European Journal of Operational Research*, **270**(1), 118–131.

Irnich, S. (2008). Solution of real-world postman problems. *European Journal of Operational Research*, **190**(1), 52–67.

Johnson, D. S., Gutin, G., McGeoch, L. A., Yeo, A., Zhang, W., and Zverovitch, A. (2007). Experimental analysis of heuristics for the ATSP. In G. Gutin and A. P. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimization*, pages 445–487. Springer US, Boston, MA.

Li, F., Golden, B., and Wasil, E. (2005). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, **32**(5), 1165–1179.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, **24**(11), 1097–1100.

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

Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 399–419. Springer.

Pop, P. C., Kara, I., and Marc, A. H. (2012). New mathematical models of the generalized vehicle routing problem and extensions. *Applied Mathematical Modelling*, **36**(1), 97–107.

Pop, P. C., Fuksz, L., Marc, A. H., and Sabo, C. (2018). A novel two-level optimization approach for clustered vehicle routing problem. *Computers & Industrial Engineering*, **115**(Supplement C), 304–318.

Ropke, S. (2009). Parallel large neighborhood search-a software framework. In *MIC 2009. The VIII Metaheuristics International Conference.*

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

Ropke, S. and Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, **171**(3), 750–775. Feature Cluster: Heuristic and Stochastic Methods in Optimization. Feature Cluster: New Opportunities for Operations Research.

Sevaux, M. and Sörensen, K. (2008). Hamiltonian paths in large clustered routing problems. In *Proceedings of the EU/MEeting 2008 workshop on Metaheuristics for Logistics and Vehicle Routing, EU/ME'08*, pages 4:1–4:7, Troyes, France.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. *Lecture Notes in Computer Science*, **1520**, 417–431.

Simonetti, N. and Balas, E. (1996). Implementation of a linear time algorithm for certain generalized traveling salesman problems. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization: 5th International IPCO Conference Vancouver, British Columbia, Canada, June 3–5, 1996 Proceedings*, pages 316–329. Springer Berlin Heidelberg, Berlin, Heidelberg.

Toth, P. and Vigo, D., editors (2014). *Vehicle Routing*, volume 18 of *MOS-SIAM Series on Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, **86**, 629–659.

Vidal, T., Battarra, M., Subramanian, A., and Erdoğan, G. (2015). Hybrid metaheuristics for the clustered vehicle routing problem. *Computers & Operations Research*, **58**, 87–99.

This appendix is supposed to become online supplementary material.

Appendix

I. Detailed Results

I.1. Detailed Results for the `GVRP` *Instances*

Detailed instance-by-instance results for the `GVRP` instance set are provided in Tables 5–8. The instance is described by the number of customers n, the number of vehicles k in the original CVRP instance, the number of clusters N, and the number of vehicles m. In addition, BKS gives the best known solution (written in bold if proven optimal) and *First found by* refers to the article (or our LMNS) that has found this solution first. For our LMNS, we show the best solution out of ten runs (Best), the average solution over ten runs (Avg.), and the average total time over ten runs T derived by setting $\text{LMNS}^{5}_{10\,000}$ (which means the LMNS is run for 10 000 iterations and with $k_{\text{post}} = 5$). If the BKS was first found by our LMNS, we omit the number of iterations in the column *First found by* for simplicity. For example, we refer to setting $\text{LMNS}^{5}_{10\,000}$ by LMNS^{5}. If it was found with both $k_{\text{post}} = 3$ and $k_{\text{post}} = 5$ we state $\text{LMNS}^{3/5}$. Furthermore, LMNS* declares a solution found during computational experiments.

Instance							$LMNS^{5}_{10\,000}$		
	n	k	N	m	BKS	*First found by*	Best	Avg.	T
A	31	5	16	2	**595**	Hintsch and Irnich (2018a)	595	606.1	1.2
A	32	5	17	3	**528**	Hintsch and Irnich (2018a)	528	528	1.7
A	32	6	17	3	**561**	Hintsch and Irnich (2018a)	561	563.1	1.6
A	33	5	17	3	**568**	Hintsch and Irnich (2018a)	568	568	1.8
A	35	5	18	2	**596**	Hintsch and Irnich (2018a)	596	596	1.6
A	36	5	19	3	**573**	Hintsch and Irnich (2018a)	573	573	2.1
A	36	6	19	3	**660**	Hintsch and Irnich (2018a)	660	660	1.4
A	37	5	19	3	**547**	Hintsch and Irnich (2018a)	547	547	2.2
A	38	5	20	3	**659**	Hintsch and Irnich (2018a)	659	659	2.1
A	38	6	20	3	**676**	Hintsch and Irnich (2018a)	676	676.7	2.1
A	43	6	22	3	**723**	Hintsch and Irnich (2018a)	723	723	2.3
A	44	6	23	4	**679**	Hintsch and Irnich (2018a)	679	679	2.5
A	44	7	23	4	**774**	Hintsch and Irnich (2018a)	774	774	1.7
A	45	7	23	4	**708**	Hintsch and Irnich (2018a)	708	709.5	2.5
A	47	7	24	4	**784**	Hintsch and Irnich (2018a)	784	784	2.1
A	52	7	27	4	**732**	Hintsch and Irnich (2018a)	732	732.6	2.8
A	53	7	27	4	**806**	Hintsch and Irnich (2018a)	806	806	3.0
A	54	9	28	5	**778**	Hintsch and Irnich (2018a)	778	778	2.2
A	59	9	30	5	**877**	Hintsch and Irnich (2018a)	877	877	2.7
A	60	9	31	5	**749**	Hintsch and Irnich (2018a)	749	749	3.7
A	61	8	31	4	**849**	Hintsch and Irnich (2018a)	849	849	4.4
A	62	9	32	5	**1043**	Hintsch and Irnich (2018a)	1043	1043	4.1
A	62	10	32	5	**895**	Hintsch and Irnich (2018a)	895	895	4.1
A	63	9	32	5	**895**	Hintsch and Irnich (2018a)	895	895.1	3.0
A	64	9	33	5	**825**	Hintsch and Irnich (2018a)	825	825.8	5.6
A	68	9	35	5	**857**	Hintsch and Irnich (2018a)	857	857	6.3
A	79	10	40	5	**1115**	Hintsch and Irnich (2018a)	1115	1115	4.4
B	30	5	16	3	**451**	Hintsch and Irnich (2018a)	451	451	1.4
B	33	5	17	3	**495**	Hintsch and Irnich (2018a)	495	495	2.1
B	34	5	18	3	**654**	Hintsch and Irnich (2018a)	654	654	1.9
B	37	6	19	3	**479**	Hintsch and Irnich (2018a)	479	479	2.0
B	38	5	20	3	**378**	Hintsch and Irnich (2018a)	378	378	1.7
B	40	6	21	3	**514**	Hintsch and Irnich (2018a)	514	514	1.9
B	42	6	22	3	**522**	Hintsch and Irnich (2018a)	522	522	2.4
B	43	7	22	4	**562**	Hintsch and Irnich (2018a)	562	562	1.8
B	44	5	23	3	**542**	Hintsch and Irnich (2018a)	542	542	2.8
B	44	6	23	4	**506**	Hintsch and Irnich (2018a)	506	506	2.5
B	49	7	25	4	**495**	Hintsch and Irnich (2018a)	495	495	3.3
B	49	8	25	5	954	Hintsch and Irnich (2018a)	954	954	2.6
B	50	7	26	4	**672**	Hintsch and Irnich (2018a)	672	672	2.9
B	51	7	26	4	**485**	Hintsch and Irnich (2018a)	485	485	3.3
B	55	7	28	4	520	Hintsch and Irnich (2018a)	520	520	3.6
B	56	7	29	4	776	$LMNS^{3/5}$	776	776	3.9
B	56	9	29	5	**983**	Hintsch and Irnich (2018a)	983	983	2.8
B	62	10	32	5	**865**	Hintsch and Irnich (2018a)	865	865	3.5
B	63	9	32	5	**550**	Hintsch and Irnich (2018a)	550	550	4.8
B	65	9	33	5	849	$LMNS^{3/5}$	849	849	3.5
B	66	10	34	5	721	$LMNS^{3/5}$	721	721	4.7
B	67	9	34	5	**745**	Hintsch and Irnich (2018a)	745	745	3.9
B	77	10	39	5	**842**	Hintsch and Irnich (2018a)	842	843.4	3.9

Table 5: Detailed results for the GVRP-2 instances, subsets A and B.

Instance							$LMNS^{5}_{10\,000}$		
	n	k	N	m	BKS	*First found by*	Best	Avg.	T
P	15	8	8	5	**299**	Hintsch and Irnich (2018a)	299	299	0.2
P	18	2	10	2	**195**	Hintsch and Irnich (2018a)	195	195	0.7
P	19	2	10	2	**208**	Hintsch and Irnich (2018a)	208	208	0.8
P	20	2	11	2	**208**	Hintsch and Irnich (2018a)	208	208	1.0
P	21	2	11	2	**209**	Hintsch and Irnich (2018a)	209	209	1.0
P	21	8	11	5	**397**	Hintsch and Irnich (2018a)	397	397	0.4
P	22	8	12	5	**369**	Hintsch and Irnich (2018a)	369	369	0.5
P	39	5	20	3	**401**	Hintsch and Irnich (2018a)	401	401	2.5
P	44	5	23	3	**443**	Hintsch and Irnich (2018a)	443	443	2.9
P	49	7	25	4	**464**	Hintsch and Irnich (2018a)	464	464.4	3.4
P	49	8	25	4	**501**	Hintsch and Irnich (2018a)	501	504	1.5
P	49	10	25	5	**512**	Hintsch and Irnich (2018a)	512	517	2.1
P	50	10	26	6	**548**	Hintsch and Irnich (2018a)	548	548	2.3
P	54	7	28	4	**477**	Hintsch and Irnich (2018a)	477	477	3.6
P	54	8	28	4	**484**	Hintsch and Irnich (2018a)	484	484.5	3.8
P	54	10	28	5	**514**	Hintsch and Irnich (2018a)	514	514	2.7
P	54	15	28	8	**684**	Hintsch and Irnich (2018a)	684	684	1.8
P	59	10	30	5	**575**	Hintsch and Irnich (2018a)	575	577	2.9
P	59	15	30	8	**700**	Hintsch and Irnich (2018a)	700	700	3.0
P	64	10	33	5	**616**	Hintsch and Irnich (2018a)	616	616	4.0
P	69	10	35	5	**643**	Hintsch and Irnich (2018a)	643	643	4.5
P	75	4	38	2	**557**	Hintsch and Irnich (2018a)	557	561.6	6.8
P	75	5	38	3	**571**	Hintsch and Irnich (2018a)	571	571	7.1
P	100	4	51	2	**645**	$LMNS^{3/5}$	645	645	16.5
G	261	25	131	12	3655	LMNS*	3668	3692.3	19.6
C	100	10	51	5	**628**	Hintsch and Irnich (2018a)	628	628	7.9
C	120	7	61	4	799	$LMNS^{3/5}$	799	806	11.9
C	150	12	76	6	805	$LMNS^{3/5}$	805	805.9	19.4
C	199	16	100	8	944	$LMNS^{3}$	948	953.7	13.8

Table 6: Detailed results for the GVRP-2 instances, subsets P and GC.

Instance							$LMNS^5_{10\,000}$		
	n	k	N	m	BKS	*First found by*	Best	Avg.	T
A	31	5	11	2	**515**	Defryn and Sörensen (2017)	515	515	1.5
A	32	5	11	2	**461**	Defryn and Sörensen (2017)	461	461	1.7
A	32	6	11	2	**554**	Defryn and Sörensen (2017)	554	554	1.7
A	33	5	12	2	**538**	Defryn and Sörensen (2017)	538	538	1.9
A	35	5	12	2	**543**	Defryn and Sörensen (2017)	543	543	1.5
A	36	5	13	2	**545**	Hintsch and Irnich (2018a)	545	545	2.1
A	36	6	13	2	**605**	Defryn and Sörensen (2017)	605	605	1.8
A	37	5	13	2	**507**	Battarra *et al.* (2014)	507	507	2.2
A	38	5	13	2	**588**	Defryn and Sörensen (2017)	588	588	2.4
A	38	6	13	2	**603**	Defryn and Sörensen (2017)	603	603	2.1
A	43	6	15	2	**691**	Defryn and Sörensen (2017)	691	691.8	2.0
A	44	6	15	3	**652**	Defryn and Sörensen (2017)	652	652	2.6
A	44	7	15	3	**661**	Defryn and Sörensen (2017)	661	661	2.1
A	45	7	16	3	**642**	Defryn and Sörensen (2017)	642	642	2.7
A	47	7	16	3	**680**	Defryn and Sörensen (2017)	680	680	2.5
A	52	7	18	3	**627**	Defryn and Sörensen (2017)	627	627	3.3
A	53	7	18	3	**699**	Defryn and Sörensen (2017)	699	699	3.5
A	54	9	19	3	**645**	Defryn and Sörensen (2017)	645	645	3.3
A	59	9	20	3	**762**	Defryn and Sörensen (2017)	762	762	3.5
A	60	9	21	4	**671**	Defryn and Sörensen (2017)	671	672.6	3.4
A	61	8	21	3	**771**	Defryn and Sörensen (2017)	771	771	4.2
A	62	10	21	4	**779**	Defryn and Sörensen (2017)	779	779	3.5
A	62	9	21	3	**837**	Defryn and Sörensen (2017)	837	837	3.3
A	63	9	22	3	**767**	Defryn and Sörensen (2017)	767	767	3.8
A	64	9	22	3	**693**	Defryn and Sörensen (2017)	693	693	3.7
A	68	9	23	3	**794**	Defryn and Sörensen (2017)	794	798	3.8
A	79	10	27	4	**944**	Defryn and Sörensen (2017)	944	944	5.1
B	30	5	11	2	**375**	Battarra *et al.* (2014)	375	375	1.6
B	33	5	12	2	**415**	Defryn and Sörensen (2017)	415	415	2.0
B	34	5	12	2	**557**	Defryn and Sörensen (2017)	557	557.3	2.1
B	37	6	13	2	**427**	Defryn and Sörensen (2017)	427	427	1.8
B	38	5	13	2	**317**	Defryn and Sörensen (2017)	317	317	2.3
B	40	6	14	2	**469**	Defryn and Sörensen (2017)	469	469	2.3
B	42	6	15	2	**405**	Defryn and Sörensen (2017)	405	405	2.6
B	43	7	15	3	**443**	Defryn and Sörensen (2017)	443	443	1.8
B	44	5	15	2	**489**	Defryn and Sörensen (2017)	489	489	2.8
B	44	6	15	2	**386**	Defryn and Sörensen (2017)	386	386	2.5
B	49	7	17	3	**464**	Defryn and Sörensen (2017)	464	464	2.9
B	49	8	17	3	**661**	Defryn and Sörensen (2017)	661	661	2.7
B	50	7	17	3	**578**	Defryn and Sörensen (2017)	578	578	3.3
B	51	7	18	3	**427**	Battarra *et al.* (2014)	427	427	3.6
B	55	7	19	3	**420**	Defryn and Sörensen (2017)	420	420	3.8
B	56	7	19	3	**622**	Defryn and Sörensen (2017)	622	622	3.5
B	56	9	19	3	**746**	Defryn and Sörensen (2017)	746	746	3.6
B	62	10	21	3	**685**	Battarra *et al.* (2014)	685	685	3.2
B	63	9	22	4	**524**	Defryn and Sörensen (2017)	524	524	4.6
B	65	9	22	3	**683**	Defryn and Sörensen (2017)	683	685.5	4.2
B	66	10	23	4	**619**	Defryn and Sörensen (2017)	619	619	4.8
B	67	9	23	3	**582**	Defryn and Sörensen (2017)	582	582	3.6
B	77	10	26	4	**704**	Defryn and Sörensen (2017)	704	704	5.4

Table 7: Detailed results for the `GVRP-3` instances, subsets `A` and `B`.

Instance							$LMNS^{5}_{10\,000}$		
	n	k	N	m	BKS	*First found by*	Best	Avg.	T
P	15	8	6	4	**251**	Defryn and Sörensen (2017)	251	251	0.4
P	18	2	7	1	**170**	Defryn and Sörensen (2017)	170	170	0.8
P	19	2	7	1	**177**	Defryn and Sörensen (2017)	177	177	0.8
P	20	2	7	1	**179**	Defryn and Sörensen (2017)	179	179	0.9
P	21	2	8	1	**183**	Defryn and Sörensen (2017)	183	183	1.0
P	21	8	8	4	**365**	Battarra *et al.* (2014)	365	365	0.5
P	22	8	8	3	**270**	Defryn and Sörensen (2017)	270	270	0.7
P	39	5	14	2	**381**	Defryn and Sörensen (2017)	381	381	2.5
P	44	5	15	2	**422**	Defryn and Sörensen (2017)	422	422	2.8
P	49	7	17	3	**430**	Defryn and Sörensen (2017)	430	430	3.1
P	49	8	17	3	**441**	Defryn and Sörensen (2017)	441	441.3	2.8
P	49	10	17	4	**471**	Defryn and Sörensen (2017)	471	471	2.8
P	50	10	17	4	**493**	Defryn and Sörensen (2017)	493	493	2.6
P	54	7	19	3	**454**	Hintsch and Irnich (2018a)	454	454.2	3.6
P	54	8	19	3	**454**	Hintsch and Irnich (2018a)	454	454.8	3.8
P	54	10	19	4	**481**	Defryn and Sörensen (2017)	481	481.4	3.1
P	54	15	19	6	**572**	Defryn and Sörensen (2017)	572	572	2.4
P	59	10	20	4	**534**	Hintsch and Irnich (2018a)	534	534.1	4.0
P	59	15	20	5	**591**	Defryn and Sörensen (2017)	591	591	2.5
P	64	10	22	4	**575**	Hintsch and Irnich (2018a)	575	575	4.7
P	69	10	24	4	**602**	Defryn and Sörensen (2017)	602	602	5.1
P	75	4	26	2	**556**	Hintsch and Irnich (2018a)	556	556	7.5
P	75	5	26	2	**556**	Defryn and Sörensen (2017)	556	556.6	7.5
P	100	4	34	2	**649**	Defryn and Sörensen (2017)	649	649	12.9
G	261	25	88	9	3178	$LMNS^{3/5}$	3178	3178	50.3
C	100	10	34	4	**598**	Defryn and Sörensen (2017)	598	599.5	9.5
C	120	7	41	3	680	$LMNS^{3/5}$	680	693.2	10.4
C	150	12	51	4	**756**	Hintsch and Irnich (2018a)	756	756	19.3
C	199	16	67	6	865	$LMNS^{3/5}$	865	865	35.3

Table 8: Detailed results for the `GVRP-3` instances, subsets `P` and `GC`.

I.2. Detailed Results for the `Golden` *Instances*

Analogous to Section I.1, detailed results for the `Golden` instances are given in Tables 9 to 12 (without the number of vehicles k in the original CVRP instance). In addition, we give the best and average solution over ten runs for setting $\mathrm{LMNS}^{5}_{10\,000}(10s)$, where the LMNS is stopped after the time limit of 10 seconds.

Instance						$LMNS^5_{10\,000}$			$LMNS^5_{10\,000}(10s)$	
	n	N	m	BKS	*First found by*	Best	Avg.	T	Best	Avg.
`Golden1`	240	17	4	**4640**	Hintsch and Irnich (2018a)	4640	4640	30	4640	4640.6
`Golden1`	240	18	4	4645	Hintsch and Irnich (2018a)	4645	4645	31	4645	4645
`Golden1`	240	19	4	4650	Hintsch and Irnich (2018a)	4650	4650	33	4650	4650
`Golden1`	240	21	4	4650	Hintsch and Irnich (2018a)	4650	4650	33	4650	4650
`Golden1`	240	22	4	4650	$LMNS^{3/5}$	4650	4650	33	4650	4650
`Golden1`	240	25	4	4650	$LMNS^{3/5}$	4650	4651.2	35	4650	4653
`Golden1`	240	27	4	4652	$LMNS^{3/5}$	4652	4652	35	4652	4652.6
`Golden1`	240	31	4	4665	$LMNS^{3/5}$	4665	4665	44	4665	4665
`Golden1`	240	35	4	4619	$LMNS^{3/5}$	4619	4619.8	46	4619	4620.8
`Golden1`	240	41	4	4619	$LMNS^{3/5}$	4619	4621.3	44	4619	4628.3
`Golden1`	240	49	4	4607	LMNS*	4619	4625.5	47	4619	4629.6
`Golden2`	320	22	4	7394	$LMNS^5$	7394	7395.9	66	7395	7400.4
`Golden2`	320	23	4	**7369**	Hintsch and Irnich (2018a)	7372	7381.2	66	7386	7398.8
`Golden2`	320	25	4	7367	$LMNS^{3/5}$	7367	7370.4	69	7367	7380.9
`Golden2`	320	27	4	7333	$LMNS^{3/5}$	7333	7334.3	72	7333	7343.1
`Golden2`	320	30	4	7329	$LMNS^{3/5}$	7329	7329	78	7329	7336.5
`Golden2`	320	33	4	7311	$LMNS^{3/5}$	7311	7314.1	80	7312	7320.3
`Golden2`	320	36	4	7293	$LMNS^{3/5}$	7293	7293.2	84	7293	7304.1
`Golden2`	320	41	4	7283	$LMNS^5$	7283	7286.2	88	7288	7296.7
`Golden2`	320	46	4	7284	$LMNS^5$	7284	7290.7	95	7291	7303.1
`Golden2`	320	54	4	7274	LMNS*	7277	7278.7	101	7282	7285.9
`Golden2`	320	65	4	7261	LMNS*	7264	7272.4	104	7281	7286.6
`Golden3`	400	27	4	10077	$LMNS^{3/5}$	10077	10078.5	107	10077	10105.6
`Golden3`	400	29	4	10018	$LMNS^{3/5}$	10018	10020.6	113	10023	10035.9
`Golden3`	400	31	4	10002	LMNS*	10003	10012.7	126	10026	10046.6
`Golden3`	400	34	4	9995	LMNS*	9999	10004	131	10007	10020.1
`Golden3`	400	37	4	9986	$LMNS^5$	9986	9999.4	131	10018	10032.3
`Golden3`	400	41	4	9926	$LMNS^{3/5}$	9926	9932.9	135	9938	9976.5
`Golden3`	400	45	4	9936	LMNS*	9946	9953.9	143	9965	9984.5
`Golden3`	400	51	4	9916	LMNS*	9921	9932.1	152	9936	9945.8
`Golden3`	400	58	4	9910	LMNS*	9926	9931	169	9930	9951.7
`Golden3`	400	67	4	9901	LMNS*	9903	9907.9	174	9941	10007.4
`Golden3`	400	81	4	9868	LMNS*	9871	9875.7	185	9884	9927.5
`Golden4`	480	33	4	12741	$LMNS^{3/5}$	12741	12749.5	179	12756	12827.7
`Golden4`	480	35	4	12740	$LMNS^3$	12741	12748.3	182	12754	12840.2
`Golden4`	480	37	4	12645	$LMNS^{3/5}$	12645	12645.8	191	12651	12715.3
`Golden4`	480	41	4	12568	$LMNS^{3/5}$	12568	12568	190	12568	12649.8
`Golden4`	480	44	4	12566	$LMNS^5$	12566	12599.4	190	12605	12687.2
`Golden4`	480	49	4	12566	LMNS*	12568	12597.4	196	12582	12702.5
`Golden4`	480	54	4	12525	$LMNS^5$	12525	12609.5	191	12583	12750.1
`Golden4`	480	61	4	12558	$LMNS^{3/5}$	12558	12558	207	12562	12585.3
`Golden4`	480	69	4	12573	LMNS*	12575	12581.1	225	12600	12655
`Golden4`	480	81	4	12555	LMNS*	12557	12580.6	270	12601	12641.5
`Golden4`	480	97	4	12528	$LMNS^{3/5}$	12528	12567.5	269	12637	12727.1
`Golden5`	200	14	4	**6970**	Hintsch and Irnich (2018a)	6970	6970	22	6970	6970
`Golden5`	200	15	3	**6742**	Hintsch and Irnich (2018a)	6742	6752	26	6742	6752
`Golden5`	200	16	3	**6742**	Hintsch and Irnich (2018a)	6742	6849.1	26	6742	6849.1
`Golden5`	200	17	3	**6862**	Hintsch and Irnich (2018a)	6862	6868	26	6862	6872.3
`Golden5`	200	19	4	**6874**	Hintsch and Irnich (2018a)	6874	6874	25	6874	6874
`Golden5`	200	21	4	**6816**	Hintsch and Irnich (2018a)	6816	6817.4	26	6816	6825.9
`Golden5`	200	23	4	**6750**	Hintsch and Irnich (2018a)	6750	6750	25	6750	6750
`Golden5`	200	26	4	**6704**	Hintsch and Irnich (2018a)	6704	6704	27	6704	6704
`Golden5`	200	29	4	**6704**	Hintsch and Irnich (2018a)	6704	6704	28	6704	6704
`Golden5`	200	34	4	**6684**	Hintsch and Irnich (2018a)	6684	6692.4	29	6684	6692.4
`Golden5`	200	41	4	**6557**	Hintsch and Irnich (2018a)	6557	6578.2	32	6557	6578.4

Table 9: Detailed results for the `Golden` instances `1-5`.

Instance						$LMNS^5_{10\,000}$			$LMNS^5_{10\,000}(10s)$	
	n	N	m	BKS	*First found by*	Best	Avg.	T	Best	Avg.
Golden6	280	19	3	**8115**	Hintsch and Irnich (2018a)	8115	8115.3	54	8115	8116.8
Golden6	280	21	3	**8119**	Hintsch and Irnich (2018a)	8119	8125.5	52	8119	8131.7
Golden6	280	22	3	**8107**	Hintsch and Irnich (2018a)	8107	8113.7	52	8107	8122
Golden6	280	24	4	**8316**	Hintsch and Irnich (2018a)	8316	8318.8	52	8316	8320.5
Golden6	280	26	4	**8249**	Hintsch and Irnich (2018a)	8249	8256.4	54	8249	8288.2
Golden6	280	29	4	8244	$LMNS^{3/5}$	8244	8251.4	60	8244	8254
Golden6	280	32	4	8179	$LMNS^{3/5}$	8179	8197.3	59	8179	8215.4
Golden6	280	36	4	8179	$LMNS^{3/5}$	8179	8180.9	59	8179	8199.1
Golden6	280	41	4	8204	$LMNS^{3/5}$	8204	8206.5	66	8204	8219.1
Golden6	280	47	4	8179	$LMNS^{3/5}$	8179	8192.6	65	8181	8200.3
Golden6	280	57	4	8204	$LMNS^{3/5}$	8204	8205.6	75	8205	8225
Golden7	360	25	3	**9318**	Hintsch and Irnich (2018a)	9318	9321.5	99	9321	9341.4
Golden7	360	26	3	**9295**	Hintsch and Irnich (2018a)	9307	9314.1	101	9313	9330
Golden7	360	28	3	9271	$LMNS^3$	9272	9282.7	109	9274	9299.3
Golden7	360	31	4	**9418**	Hintsch and Irnich (2018a)	9418	9442.6	101	9451	9458.5
Golden7	360	33	4	9395	LMNS*	9401	9401.8	103	9401	9404.4
Golden7	360	37	4	**9395**	Hintsch and Irnich (2018a)	9395	9403.7	104	9395	9427.1
Golden7	360	41	4	9386	$LMNS^5$	9386	9400.3	108	9386	9414.5
Golden7	360	46	4	9368	$LMNS^{3/5}$	9368	9376.7	102	9383	9391
Golden7	360	52	4	9365	$LMNS^{3/5}$	9365	9373.1	114	9375	9411.4
Golden7	360	61	4	9316	$LMNS^{3/5}$	9316	9343.6	128	9343	9369.4
Golden7	360	73	4	9302	$LMNS^5$	9302	9314.9	145	9325	9368.8
Golden8	440	30	4	10409	$LMNS^5$	10409	10417.1	133	10415	10464.8
Golden8	440	32	4	10409	LMNS*	10411	10422.3	134	10420	10442.9
Golden8	440	34	4	10409	LMNS*	10411	10418.3	139	10424	10451.7
Golden8	440	37	4	10360	LMNS*	10368	10378.9	146	10386	10410.1
Golden8	440	41	4	10360	LMNS*	10368	10371.2	152	10379	10424.9
Golden8	440	45	4	10385	LMNS*	10387	10392.4	152	10393	10438.7
Golden8	440	49	4	10399	$LMNS^5$	10399	10413.2	165	10425	10454
Golden8	440	56	4	10371	$LMNS^{3/5}$	10371	10393.8	180	10412	10443.7
Golden8	440	63	4	10361	LMNS*	10365	10391	184	10413	10451.3
Golden8	440	74	4	10356	LMNS*	10363	10368.6	201	10397	10455.3
Golden8	440	89	4	10281	$LMNS^3$	10282	10292.1	217	10352	10419.1
Golden9	255	18	4	**281**	Hintsch and Irnich (2018a)	281	281	39	281	282.1
Golden9	255	19	4	**279**	Hintsch and Irnich (2018a)	279	279.2	38	279	280.2
Golden9	255	20	4	**276**	Hintsch and Irnich (2018a)	276	276.6	40	276	277.4
Golden9	255	22	4	**276**	Hintsch and Irnich (2018a)	276	276.7	44	277	277.1
Golden9	255	24	4	**276**	Hintsch and Irnich (2018a)	276	276.9	44	277	277.3
Golden9	255	26	4	**273**	Hintsch and Irnich (2018a)	273	273.9	46	274	274.3
Golden9	255	29	4	**273**	Hintsch and Irnich (2018a)	273	273.6	45	273	274.2
Golden9	255	32	4	**273**	Hintsch and Irnich (2018a)	273	273.9	48	274	274.3
Golden9	255	37	4	**273**	Hintsch and Irnich (2018a)	273	273.9	50	274	274.6
Golden9	255	43	4	**270**	$LMNS^{3/5}$	270	270.8	53	271	272
Golden9	255	52	4	269	$LMNS^{3/5}$	269	269	57	269	269.7
Golden10	323	22	4	**346**	Hintsch and Irnich (2018a)	346	347	63	347	347.6
Golden10	323	24	4	**346**	Hintsch and Irnich (2018a)	346	346.2	65	346	346.9
Golden10	323	25	4	**346**	Hintsch and Irnich (2018a)	346	346.2	65	346	347
Golden10	323	27	4	**346**	Hintsch and Irnich (2018a)	346	346.2	68	346	346.7
Golden10	323	30	4	**347**	Hintsch and Irnich (2018a)	347	348	71	348	349
Golden10	323	33	4	**344**	Hintsch and Irnich (2018a)	344	344.1	73	344	345
Golden10	323	36	4	**344**	Hintsch and Irnich (2018a)	344	344.1	72	344	345.7
Golden10	323	41	4	346	$LMNS^{3/5}$	346	346	79	346	346.9
Golden10	323	47	4	344	$LMNS^{3/5}$	344	345.1	83	346	346.7
Golden10	323	54	4	340	$LMNS^5$	340	341.1	82	341	343.4
Golden10	323	65	4	335	$LMNS^{3/5}$	335	337.1	87	338	339.8

Table 10: Detailed results for the Golden instances 6-10.

Instance						$LMNS^5_{10\,000}$			$LMNS^5_{10\,000}(10s)$	
	n	N	m	BKS	*First found by*	Best	Avg.	T	Best	Avg.
Golden11	399	27	5	**434**	Hintsch and Irnich (2018a)	434	434.7	90	435	436.3
Golden11	399	29	5	**434**	Hintsch and Irnich (2018a)	434	434.4	97	436	436.6
Golden11	399	31	5	**433**	Hintsch and Irnich (2018a)	435	435.6	95	436	437.3
Golden11	399	34	5	**427**	Hintsch and Irnich (2018a)	428	429.2	101	430	431
Golden11	399	37	5	**427**	LMNS*	428	429.1	99	429	430.5
Golden11	399	40	5	**425**	LMNS*	426	427.1	108	428	428.8
Golden11	399	45	5	**425**	$LMNS^{3/5}$	425	425.3	112	426	427.8
Golden11	399	50	5	423	LMNS*	424	425.8	109	427	428.3
Golden11	399	58	5	422	$LMNS^{3/5}$	422	423.6	123	425	426.3
Golden11	399	67	5	422	$LMNS^5$	422	423.6	130	425	426.3
Golden11	399	80	5	417	$LMNS^{3/5}$	417	417.6	138	420	421.5
Golden12	483	33	5	512	$LMNS^{3/5}$	512	513.1	138	514	515.9
Golden12	483	35	5	512	$LMNS^{3/5}$	512	512.2	139	513	515.3
Golden12	483	38	5	511	LMNS*	513	513	146	513	514.3
Golden12	483	41	5	512	LMNS*	513	513.4	145	515	516.2
Golden12	483	44	5	511	LMNS*	512	512.8	151	516	516.8
Golden12	483	49	5	511	LMNS*	512	513.2	163	515	516.5
Golden12	483	54	5	510	$LMNS^5$	510	513.1	164	514	517.6
Golden12	483	61	5	510	LMNS*	512	512.6	181	514	516.4
Golden12	483	70	5	509	$LMNS^{3/5}$	509	509.8	185	511	515.8
Golden12	483	81	5	502	$LMNS^5$	502	504.1	209	508	510.6
Golden12	483	97	5	502	LMNS*	504	505	235	505	513.1
Golden13	252	17	4	**530**	Hintsch and Irnich (2018a)	530	530.4	40	530	530.7
Golden13	252	19	4	**521**	Hintsch and Irnich (2018a)	521	521.8	40	521	521.8
Golden13	252	20	4	**521**	Hintsch and Irnich (2018a)	521	521.5	42	521	521.8
Golden13	252	22	4	**523**	Hintsch and Irnich (2018a)	523	523.2	42	523	523.9
Golden13	252	23	4	**523**	Hintsch and Irnich (2018a)	523	523.2	43	523	523.5
Golden13	252	26	4	**523**	Hintsch and Irnich (2018a)	523	523	46	523	523.2
Golden13	252	29	4	**522**	Hintsch and Irnich (2018a)	522	522	48	522	522.8
Golden13	252	32	4	**521**	Hintsch and Irnich (2018a)	521	521.2	49	521	522.1
Golden13	252	37	4	**521**	Hintsch and Irnich (2018a)	521	521.9	53	522	522.5
Golden13	252	43	4	**521**	Hintsch and Irnich (2018a)	521	521	54	521	521.3
Golden13	252	51	4	**521**	$LMNS^{3/5}$	521	521	58	521	521.3
Golden14	320	22	4	**665**	Hintsch and Irnich (2018a)	666	666	62	666	666.9
Golden14	320	23	4	**662**	Hintsch and Irnich (2018a)	662	662	64	662	662.2
Golden14	320	25	4	**660**	Hintsch and Irnich (2018a)	660	660	66	660	660.7
Golden14	320	27	4	**660**	Hintsch and Irnich (2018a)	660	660	67	660	660.2
Golden14	320	30	4	**660**	Hintsch and Irnich (2018a)	660	660	69	660	660.1
Golden14	320	33	4	**660**	$LMNS^{3/5}$	660	660	71	660	660.3
Golden14	320	36	4	**658**	$LMNS^{3/5}$	658	658.9	75	658	660.2
Golden14	320	41	4	**658**	Hintsch and Irnich (2018a)	658	658	82	658	658.6
Golden14	320	46	4	658	$LMNS^{3/5}$	658	659.4	87	658	659.8
Golden14	320	54	4	658	$LMNS^{3/5}$	658	659	93	659	660.4
Golden14	320	65	4	658	$LMNS^{3/5}$	658	658.2	99	658	660.2
Golden15	396	27	4	**815**	$LMNS^{3/5}$	815	816.6	94	816	817.9
Golden15	396	29	4	**815**	LMNS*	816	817.6	100	819	819.5
Golden15	396	31	4	**813**	Hintsch and Irnich (2018a)	813	814.4	101	815	817.1
Golden15	396	34	4	**813**	LMNS*	815	815.2	102	817	817.2
Golden15	396	37	4	815	$LMNS^{3/5}$	815	815.2	102	815	816.6
Golden15	396	40	4	815	$LMNS^{3/5}$	815	815.8	109	817	818
Golden15	396	45	5	817	$LMNS^{3/5}$	817	818.6	115	819	821.6
Golden15	396	50	5	815	LMNS*	819	819.2	123	821	822.2
Golden15	396	57	5	815	LMNS*	817	817.8	131	819	821.5
Golden15	396	67	5	815	LMNS*	817	817.2	142	819	820.6
Golden15	396	80	5	815	LMNS*	817	817.8	157	819	821.2

Table 11: Detailed results for the Golden instances 11-15.

Instance						$LMNS^5_{10\,000}$			$LMNS^5_{10\,000}(10s)$	
	n	N	m	BKS	*First found by*	Best	Avg.	T	Best	Avg.
Golden16	480	33	5	993	$LMNS^5$	993	995	141	997	998.7
Golden16	480	35	5	**993**	$LMNS^{3/5}$	993	994.6	144	997	997.9
Golden16	480	37	5	993	$LMNS^{3/5}$	993	994.6	150	997	999.3
Golden16	480	41	5	993	LMNS*	995	996.2	158	997	999.9
Golden16	480	44	5	993	LMNS*	995	996.2	164	998	999.7
Golden16	480	49	5	989	LMNS*	991	992.1	171	993	995.1
Golden16	480	54	5	985	$LMNS^{3/5}$	985	986	179	990	991.6
Golden16	480	61	5	985	LMNS*	987	988	193	990	991.5
Golden16	480	69	5	984	LMNS*	985	986.3	214	990	992.2
Golden16	480	81	5	984	$LMNS^5$	984	986.4	230	987	990.9
Golden16	480	97	5	984	$LMNS^3$	985	985.6	247	990	992
Golden17	240	17	3	**386**	Hintsch and Irnich (2018a)	386	386	44	386	386
Golden17	240	18	3	**385**	Hintsch and Irnich (2018a)	385	385	45	385	385
Golden17	240	19	3	**385**	Hintsch and Irnich (2018a)	385	385	46	385	385.1
Golden17	240	21	3	**385**	Hintsch and Irnich (2018a)	385	385	47	385	385
Golden17	240	22	3	**385**	Hintsch and Irnich (2018a)	385	385	47	385	385
Golden17	240	25	3	**382**	Hintsch and Irnich (2018a)	382	382.2	47	382	382.3
Golden17	240	27	3	**382**	Hintsch and Irnich (2018a)	382	382	49	382	382.1
Golden17	240	31	4	**390**	Hintsch and Irnich (2018a)	390	390	51	390	390.3
Golden17	240	35	4	390	$LMNS^{3/5}$	390	390	57	390	390.3
Golden17	240	41	4	**388**	Hintsch and Irnich (2018a)	388	388.4	59	388	389.3
Golden17	240	49	4	387	$LMNS^{3/5}$	387	387.2	60	387	387.9
Golden18	300	21	4	**558**	Hintsch and Irnich (2018a)	558	558	58	558	558.2
Golden18	300	22	4	**558**	Hintsch and Irnich (2018a)	558	558	59	558	558.2
Golden18	300	24	4	**558**	Hintsch and Irnich (2018a)	558	558	64	558	558.1
Golden18	300	26	4	**562**	Hintsch and Irnich (2018a)	562	562	63	562	562.6
Golden18	300	28	4	**558**	Hintsch and Irnich (2018a)	558	558	66	558	558
Golden18	300	31	4	**554**	Hintsch and Irnich (2018a)	554	554	71	554	554.5
Golden18	300	34	4	**554**	Hintsch and Irnich (2018a)	554	554.1	70	554	555.2
Golden18	300	38	4	**555**	Hintsch and Irnich (2018a)	555	555.1	74	555	556.2
Golden18	300	43	4	558	$LMNS^{3/5}$	558	558	83	558	559.2
Golden18	300	51	4	555	$LMNS^5$	555	555.9	83	558	559.5
Golden18	300	61	4	556	$LMNS^{3/5}$	556	556.6	92	557	558.4
Golden19	360	25	10	**886**	Hintsch and Irnich (2018a)	887	887.9	50	888	888.6
Golden19	360	26	10	**888**	Hintsch and Irnich (2018a)	889	889	51	889	889.6
Golden19	360	28	4	**741**	Hintsch and Irnich (2018a)	741	742	77	742	743.3
Golden19	360	31	4	**735**	Hintsch and Irnich (2018a)	737	737.5	84	739	739.2
Golden19	360	33	4	**727**	Hintsch and Irnich (2018a)	728	729.1	89	730	731
Golden19	360	37	5	**732**	Hintsch and Irnich (2018a)	733	733.5	100	734	735.1
Golden19	360	41	5	**730**	Hintsch and Irnich (2018a)	730	730.7	109	731	732.2
Golden19	360	46	5	730	$LMNS^{3/5}$	730	730.7	115	731	732.5
Golden19	360	52	5	**730**	Hintsch and Irnich (2018a)	730	730.8	120	731	733
Golden19	360	61	5	737	$LMNS^{3/5}$	737	738.5	120	740	742.4
Golden19	360	73	5	736	$LMNS^{3/5}$	736	736.9	135	739	740.4
Golden20	420	29	11	**1170**	Hintsch and Irnich (2018a)	1170	1170.9	75	1171	1171.8
Golden20	420	31	12	**1183**	Hintsch and Irnich (2018a)	1184	1184.2	74	1185	1185.9
Golden20	420	33	12	**1175**	Hintsch and Irnich (2018a)	1176	1177.1	78	1176	1178.2
Golden20	420	36	5	**1005**	LMNS*	1006	1007.2	102	1010	1012.4
Golden20	420	39	5	991	$LMNS^5$	991	992.5	110	994	998.7
Golden20	420	43	5	990	$LMNS^{3/5}$	990	990.4	115	991	993.7
Golden20	420	47	5	988	$LMNS^{3/5}$	988	989.2	121	990	991.8
Golden20	420	53	5	988	$LMNS^{3/5}$	988	988.9	125	990	993.2
Golden20	420	61	5	987	$LMNS^{3/5}$	987	988.8	133	990	992
Golden20	420	71	5	986	$LMNS^{3/5}$	986	987.4	126	988	991.7
Golden20	420	85	5	980	$LMNS^{3/5}$	980	981.2	175	982	986.8

Table 12: Detailed results for the Golden instances 16-20.

I.3. Detailed Results for the `Li` Instances

Analogous to Section I.1, detailed results for the `Li` instances are given in Table 13 (without the number of vehicles k in the original CVRP instance).

Instance						$\text{LMNS}^5_{10\,000}$		
	n	N	m	BKS	*First found by*	Best	Avg.	T
`Li`	560	113	39	27225	LMNS^5	27225	27274.9	188
`Li`	600	121	62	28759	LMNS^3	28804	28821.5	211
`Li`	640	129	10	19797	LMNS^3	19802	19832.9	208
`Li`	720	145	11	22879	LMNS^5	22879	22908.1	309
`Li`	760	153	78	35048	LMNS^3	35078	35111.6	337
`Li`	800	161	11	25423	LMNS^5	25423	25453.2	552
`Li`	840	169	86	37775	LMNS^3	37789	37825.2	413
`Li`	880	177	11	28232	LMNS^3	28240	28356.3	559
`Li`	960	193	11	30607	LMNS^3	30611	30808.2	976
`Li`	1040	209	11	33506	LMNS^3	33518	33580	1077
`Li`	1120	225	11	36219	LMNS^5	36219	36510.2	1410
`Li`	1200	241	11	38785	LMNS^5	38785	38961.4	1915

Table 13: Detailed results for the `Li` instances.