Large Multiple Neighborhood Search for the Clustered Vehicle-Routing Problem

Timo Hintsch[*,a], Stefan Irnich[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 *clustered vehicle-routing problem* (CluVRP) is a variant of the classical *capacitated vehicle-routing problem* (CVRP) in which customers are partitioned into clusters, and it is assumed that each cluster must have been served completely before the next cluster is served. This decomposes the problem into three subproblems, i.e., the assignment of clusters to routes, the routing inside each cluster, and the sequencing of the clusters in the routes. The second task requires the solution of several Hamiltonian path problems, one for each possibility to route through the cluster. We pre-compute the Hamiltonian paths for every pair of customers of each cluster. We present a *large multiple neighborhood search* (LMNS) which makes use of multiple cluster destroy and repair operators and a variable-neighborhood descent (VND) for post-optimization. The VND is based on classical neighborhoods such as relocate, 2-opt, and swap all working on the cluster level and a generalization of the Balas-Simonetti neighborhood modifying simultaneously the intra-cluster routings and the sequence of clusters in a route. Computational results with our new approach compare favorably to existing approaches from the literature.

Key words: Vehicle Routing, Clustered Vehicle Routing, Large Neighborhood Search

1. Introduction

The *clustered vehicle-routing problem* (CluVRP) generalizes the classical *capacitated vehicle-routing problem* (CVRP). The customers in the CluVRP are grouped into disjoint clusters, and the only additional constraint compared to the CVRP is that all customers of a cluster must be served by the same vehicle in consecutive visits. It means that if one customer of a cluster is served by a vehicle then all other customers of the same cluster are served by the same vehicle. Moreover, there is no customer from another cluster visited in between two customers of the same cluster.

In this paper, we present a new metaheuristic approach for the CluVRP based on the *large neighborhood search* principle (LNS, Shaw, 1998; Ropke and Pisinger, 2006b). The novelty of our *large multiple neighborhood search* (LMNS) approach is the combination of multiple destroy and repair operators in the LNS together with several neighborhoods in the local search phase. One new neighborhood search operator generalizes the Balas-Simonetti neighborhood (Balas, 1999; Balas and Simonetti, 2001) originally invented as an exponentially-sized neighborhood of the *asymmetric traveling salesman problem* (ATSP). The Balas-Simonetti neighborhood can be searched for a best-improving neighbor in polynomial time. We exploit the cluster structure of the CluVRP in several ways: First, high-quality routings through a cluster are pre-computed as ATSP solutions. An ATSP instance results from an entry-exit combination of a cluster and high-quality solutions are found with a metaheuristic combining iterated local search (ILS) and variable neighborhood descent (VND) for ATSPs. Second, the actual LMNS operates on a meta-representation of

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

the CluVRP with nodes for the depot and meta-nodes for the different clusters. This allows us to use standard CVRP neighborhoods, e.g., 2-opt, relocate, and swap to modify the grouping and sequence of clusters. Moreover, we can determine for a given sequence of clusters a best routing through the clusters efficiently using a dynamic programming (DP) model. Third, the generalized Balas-Simonetti neighborhood is used to optimize single routes of a CluVRP solution. The search operator simultaneously decides on the best permutation of a route's clusters and the routing through each cluster.

In the literature, the CluVRP is defined as a symmetric vehicle-routing problem and can therefore be modeled using a complete undirected graph $G = (V, E)$ with nodes V and edges E. The nodes comprise the set $V \setminus \{0\} = \{1, \dots, n\}$ that represents the n customers and the node 0 for the depot, where a homogeneous fleet of m vehicles is housed. The capacity of a vehicle is denoted by Q. The customers are partitioned into N *clusters* $V_1, V_2, \dots, V_{N-1}, V_N$. For the sake of convenience, we also define the *depot cluster* as $V_0 = \{0\}$. All customer cluster V_h, $h \in \{1, 2, \dots, N\}$, have a positive demand d_h and cardinality $\lambda_h = |V_h|$. All edges $\{i, j\} \in E$ have associated routing costs c_{ij}.

The task is to determine a set of m feasible routes with minimum total routing costs serving each customer exactly once. A route is feasible if

(i) it starts and ends at the depot 0,
(ii) it respects the clustering, meaning that if customer $i \in V_h$ is visited then all other customers in $V_h \setminus \{i\}$ are visited directly before or after i without any other intermediate customers, and
(iii) the demand of the visited clusters does not exceed the vehicle capacity.

If all clusters are singletons $V_h = \{h\}$, the CluVRP reduces to the CVRP. This also shows that the CluVRP is $\mathcal{NP}$-hard. For $m = 1$, the resulting problem is the *clustered traveling salesman problem* (Chisman, 1975).

The paper is structured as follows. Section 2 reviews heuristic and exact CluVRP approaches from the literature. In Section 3, we present the overall LMNS algorithm and explain details of its components. The algorithmic components of the LMNS and their interplay are carefully analyzed in Section 4. Here, we also compare our computational results with the state-of-the-art metaheuristics for the CluVRP. Final conclusions are drawn in Section 5.

2. Literature Review

The CluVRP was introduced by Sevaux and Sörensen (2008) in the context of courier companies delivering parcels to a large number of customers. In this application, the customers are divided into regional zones, and parcels are sorted into containers according to their postal code. Hence, the regional zones imply customer clusters.

To the best of our knowledge, the literature describes only two exact approaches for the CluVRP. Pop *et al.* (2012) presented two compact formulations, but did not show computational results. Battarra *et al.* (2014) developed two exact algorithms and provided results for a set of benchmark instances. Their branch-and-cut algorithm relies on a preprocessing algorithm that calculates a shortest Hamiltonian path (SHP) inside every cluster for every pair of nodes belonging to that cluster. It outperforms their branch-and-cut-and-price algorithm, which is actually a branch-and-bound with combined row-and-column generation (not based on a formulation with route variables).

Several heuristic approaches have been published. Barthélemy *et al.* (2010) transformed the problem into the CVRP by adding a large value M to all inter-cluster edges, and the transformed problem is then solved by a simulated-annealing algorithm. Defryn and Sörensen (2015) and Expósito-Izquierdo *et al.* (2016) presented different two-level approaches with two types of subproblems: The low-level routing problem changes the sequence of the customers inside each cluster. In contrast, the high-level routing problem only alters the sequence of the clusters, which imposes a CVRP that uses the clusters as its customers. Defryn and Sörensen (2015) suggested two variable neighborhood searches, one for each level. Expósito-Izquierdo *et al.* (2016) solved the high-level routing problem with the record-to-record travel algorithm of Golden *et al.* (1998). For solving the low-level problem, a mixed integer linear programming model, the construction algorithm of Christofides (1970), and the Lin-Kernighan improvement heuristic (Lin and Kernighan, 1973) are employed as exact and heuristic techniques.

A hybrid approach combining a genetic algorithm with a simulated annealing algorithm to calculate all SHPs inside the clusters was presented by Marc *et al.* (2015). Vidal *et al.* (2015) proposed two iterated local searches and a hybrid genetic algorithm called unified hybrid genetic search (UHGS). UHGS is based on the work (Vidal *et al.*, 2012) and uses the preprocessing of the SHPs per cluster (as in Battarra *et al.*, 2014) and a very efficient exploration of large neighborhoods. Vidal *et al.* (2015) produced solutions of impressive quality on an older benchmark set and generated some new large-scale CluVRP instances for which they presented the first results.

3. LMNS for the CluVRP

In this section, we describe the components of our metaheuristic used for solving the CluVRP. We start with the pre-computation of intra-cluster routes in Section 3.1, followed by the description of the new Balas-Simonetti neighborhood for clusters described in Section 3.2. Neighborhoods that allow the exchange of clusters between different routes are presented in Section 3.3. The destroy and repair operator of the LMNS and the overall algorithm are described in Sections 3.4 and 3.5.

3.1. Intra-Cluster Route Pre-Computation

We pre-compute all intra-cluster routes by heuristically solving one SHP problem for each customer cluster V_h and each entry-exit combination $(e_h, f_h) \in V_h \times V_h$ with $e_h \neq f_h$. Such a Hamiltonian path $x_h(e_h, f_h) = (e_h, \dots, f_h)$ starts at the chosen entry e_h, ends at the chosen exit f_h, and visits all remaining nodes of cluster V_h in between. The cost of the computed Hamiltonian path is denoted by $\hat{c}_{e_h f_h}$. In the symmetric case, $\sum_{h=1}^{N} \binom{\lambda_h}{2}$ Hamiltonian paths have to be calculated in total. For convenience, we define $\hat{c}_{e_h f_h} = 0$ for clusters consisting of a single node, i.e., $\lambda_h = 1$ and hence $e_h = f_h$. Moreover, we set $\hat{c}_{e_h f_h} = M$ using a large number $M > 0$ if entry and exit are identical ($e_h = f_h$), and the cluster consists of more than one customer ($\lambda_h > 1$).

We transform the SHP into a traveling salesman problem (TSP) defined by all nodes V_h of the cluster. The induced distance matrix is derived from the cost matrix (c_{ij}). The only modification is the addition of the value $-M$ to the edge $\{e_h, f_h\}$ for the entry-exit-combination (e_h, f_h) under consideration. This TSP is then solved heuristically using the Balas-Simonetti neighborhood (described in Section 3.1.1) and a combined ILS/VND (see Section 3.1.2).

3.1.1. Balas-Simonetti Neighborhood

The Balas-Simonetti neighborhood was introduced by Balas (1999), it is in fact a family of ATSP neighborhoods $\mathcal{N}_k^{BS}$ for $k \geq 2$, each of exponential size that can however be searched efficiently. Balas and Simonetti (2001) analyzed the performance of $\mathcal{N}_k^{BS}$-based improvement heuristics for the ATSP and the ATSP with time windows. We use this neighborhood as one component in an ATSP heuristic similar to the algorithm described in (Irnich, 2008). In addition, we present a generalization of the Balas-Simonetti neighborhood for the CluVRP that permutes clusters and simultaneously chooses optimal entry-exit combinations for the clusters in Section 3.2.

For describing the elements of the neighborhood $\mathcal{N}_k^{BS}$ in the ATSP, we assume that the parameter $k \geq 2$ is given. Let $x = (x_0, x_1, x_2, \dots, x_n, x_{n+1})$ be an ATSP tour or a Hamiltonian path. Each $x' = (x_{\pi(0)}, x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)}, x_{\pi(n+1)})$ is a neighbor of x in $\mathcal{N}_k^{BS}$, if the permutation π of $\{0, 1, \dots, n, n+1\}$ fulfills $\pi(0) = 0$, $\pi(n+1) = n+1$, and the following condition: if $i + k \leq j$ for a pair of indices $i, j \in \{1, 2, \dots, n\}$, then $\pi(i) \leq \pi(j)$ must hold. It means that if a node x_i that precedes another node x_j by at least k positions in the original tour x, then x_i must also precede x_j in the neighbor tour x'. In this case, we write $x' \in \mathcal{N}_k^{BS}(x)$.

A best neighbor solution $x' \in \mathcal{N}_k^{BS}(x)$ can be determined by solving a shortest-path problem in an auxiliary graph G_k^*. An example of such an auxiliary graph is shown in Figure 1 for $k = 3$ and an original tour $x = (x_0, x_1, \dots, x_n, x_{n+1})$ with $n = 5$. The auxiliary graph G_k^* is structured as follows: there are $n + 2$ identical stages, each stage has $(k + 1)2^{k-2}$ states denoted by W_i for $0 \leq i \leq n + 1$. Moreover, only states of consecutive stages i and $i + 1$ are connected via arcs. The number of arcs in $G_k^*[W_i \cup W_{i+1}]$ does not exceed

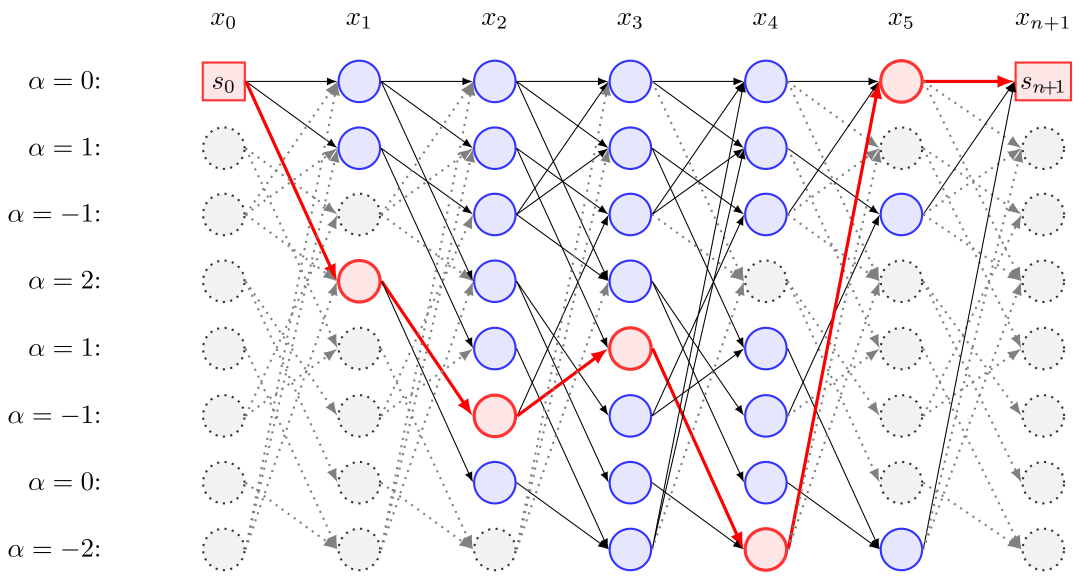

Figure 1: Auxiliary graph G_k^* for $k = 3$, current solution $x = (x_0, x_1, x_2, x_3, x_4, x_5, x_{n+1})$, and neighbor $x' = (x_0, x_3, x_1, x_4, x_2, x_5, x_{n+1}) \in \mathcal{N}_3^{BS}(x)$ implied by the highlighted s_0-s_{n+1}-path
.

$k(k+1)2^{k-2}$. Stage 1 contains the start state s_0 and stage $n+1$ the sink state s_{n+1}. Every s_0-s_{n+1}-path in G_k^* represents a neighbor x' of x, and vice versa. The idea of Balas (1999) was that each state $s \in W_i$ refers to a restricted permutation of the nodes around position i. We use a value $\alpha(s)$ to partially characterize this permutation in the sense that the state $s \in W_i$ in stage i determines the permuted node $x'_i = x_{i+\alpha(s)}$ at position i in the neighbor tour x'. Thus, the number $\alpha(s)$ is an integer strictly between $-k$ and k associated with state s. If the s_0-s_{n+1}-path contains state s at stage i, it means that node $x_{i+\alpha(s)}$ is shifted from position $i + \alpha(s)$ in the given tour x to position i in the neighbor tour x'. In Figure 1, the neighbor $x' = (x_0, x_3, x_1, x_4, x_2, x_5, x_{n+1}) \in \mathcal{N}_3^{BS}(x_0, x_1, x_2, x_3, x_4, x_5, x_{n+1})$ is represented by the highlighted path depicted by red/bold nodes and arcs.

The construction of the subgraphs $G_k^*[W_i \cup W_{i+1}]$ is nontrivial for general $k \geq 2$. Balas and Simonetti (2001) and Simonetti and Balas (1996) describe the rules that determine the arc set and the values $\alpha(s)$ for states $s \in W_i$.

A tailored dynamic programming labeling algorithm can be used to solve the shortest s_0-s_{n+1}-path problem in the auxiliary graph G_k^*. Note first that G_k^* is acyclic so that a pulling or reaching-based labeling algorithm is applicable. Second, all induced subgraphs $G_k^*[W_i \cup W_{i+1}]$ for $i \in \{0, 1, \dots, n\}$ are identical. As a consequence, only one such copy needs to be constructed beforehand, and only once. Herewith, the auxiliary graph is represented implicitly. Note also that auxiliary graphs for decreasing values of k are subgraphs. Indeed, for any $k \leq k^{\max}$, G_k^* is the subgraph of $G_{k^{\max}}^*$ induced by the first $(k+1)2^{k-2}$ states. Consequently, only $G_{k^{\max}}^*[W_i \cup W_{i+1}]$ has to be constructed and stored. Third, states that point to a position $i + \alpha(s) < 0$ or $i + \alpha(s) > n+1$ are unreachable. Moreover, those states that cannot be reached from s_0 or that cannot reach s_{n+1} are also unreachable (depicted in gray and dotted in Figure 1). Finally, the structure of G_k^* does not depend on the current solution x. Only the costs of the arcs of G_k^* depend on x: an arc $(s, s') \in W_i \times W_{i+1}$ receives the cost $c_{x_{i+\alpha(s)}, x_{i+1+\alpha(s')}}$ so that the cost of any s_0-s_{n+1}-path is identical with the cost of the resulting neighbor x'. For example, the first bold arc in Figure 1 has cost $c_{x_{0+0}, x_{1+2}} = c_{x_0, x_3}$, the second has cost $c_{x_{1+2}, x_{2-1}} = c_{x_3, x_1}$, etc.

The DP algorithm to determine a best neighbor solution can be implemented requiring $\mathcal{O}(nk^2 2^k)$ time and space. In particular, searching this exponentially sized neighborhood ($> (k/e)^{n-1}$ neighbors for $n >$

$k(k+1)$, see Gutin *et al.*, 2002, p. 233) requires only linear effort in n, i.e., the length of the ATSP tour. Furthermore, the DP is exact, i.e., determines an optimal ATSP solution when $k \geq n$. We will use this for small-sized ATSPs/SHPs. However, this is not a viable approach in general because the computational effort grows exponentially with k.

3.1.2. Overall ATSP Heuristic

We employ a mixed strategy for solving SHPs depending on the size λ_h of the hth cluster V_h. The following three parameters have to be chosen: (i) the maximum size λ_{BS} of a small cluster; (ii) the parameter k_{ATSP} of the Balas-Simonetti neighborhood used for searching large clusters, and (iii) the number It_{ATSP} of ILS iterations for large clusters.

For clusters of size $\lambda_h < 4$ there is nothing to do. If the cluster is small, i.e., $4 \leq \lambda_h \leq \lambda_{\text{BS}}$, we calculate an exact SHP for all its intra-cluster routes with the Balas-Simonetti neighborhood search. For this purpose, we set $k = \lambda_h - 2$. Then, for a given entry-exit combination (e_h, f_h), we construct an arbitrary starting solution $x_h(e_h, f_h)$ and perform a single search for a best neighbor within $\mathcal{N}^{BS}_{\lambda_h - 2}(x_h(e_h, f_h))$. This neighbor is already an optimal solution to the SHP.

Otherwise, for larger clusters with $\lambda_h > \lambda_{\text{BS}}$, we run an ILS-based heuristic (Johnson *et al.*, 2007), similar to the one described in (Irnich, 2008). First, a starting solution is constructed by the nearest neighbor heuristic. Second, some classical edge-exchange neighborhoods (we use 2-opt, Or-opt, and double-bridge, see, e.g., Funke *et al.*, 2005) and the Balas-Simonetti neighborhood with k_{ATSP} are combined within a VND (Hansen and Mladenović, 2001). Note that all three edge-exchange neighborhoods can be searched efficiently in $\mathcal{O}\left({\lambda_h}^2\right)$ time and space (see Glover, 1996). This results in a local optimum w.r.t. all four neighborhoods. Third, local optima are perturbed by two randomly chosen double-bridge moves. This creates the new starting solution for the next VND iteration. Overall, we perform It_{ATSP} iterations.

3.2. Balas-Simonetti Neighborhood for Clusters

The goal of this section is the introduction of a generalization of the Balas-Simonetti neighborhood applicable to a single CluVRP route. As before, the new family of neighborhoods is parametrized by k. For a fixed $k \geq 1$, the neighborhood allows the permutation of clusters in the same way in which nodes can be permuted in the ATSP neighborhood $\mathcal{N}^{BS}_k$. The new neighborhood is therefore of exponential size w.r.t. the number of clusters visited in the CluVRP route under consideration. Moreover, the new neighborhood allows to arbitrarily modify all entry and exit nodes for every visited cluster. This is the part of the neighborhood definition that is specific for the CluVRP. Already with two options for entry and exit per cluster, there are exponentially many neighbor routes with an identical sequence of clusters. The new neighborhood combines both the limited permutation of clusters and the choice of entry-exit combinations. We will show that still an optimal combination, i.e., a best neighbor route, can be determined efficiently.

We start with the formal description of an arbitrary CluVRP route. Such a route is denoted by $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_p, \sigma_{p+1})$ with triplets $\sigma_i = (e_i, V_{\ell_i}, f_i)$ where V_{ℓ_i} is the ith visited cluster with index ℓ_i. Herein, ℓ is a mapping from the set $\{0, 1, \ldots, p+1\}$ of positions to the set $\{0, 1, 2, \ldots, N\}$ of cluster indices. Moreover, $e_i, f_i \in V_{\ell_i}$ are the entry and exit nodes, respectively, for all $i \in \{0, 1, \ldots, p+1\}$. We assume that every route starts and ends at the depot requiring $\ell_0 = \ell_{p+1} = 0$ so that the first triplet and the last triplet are $(0, V_0, 0)$. Recall that the depot's cluster V_0 is $\{0\}$. The remaining triplets describe the routing through the customer clusters in the sense that in the ith step the cluster V_{ℓ_i} is entered at e_i, exited at f_i, and all nodes of the cluster are visited along the pre-computed Hamiltonian e_i-f_i-path with cost $\hat{c}_{e_i f_i}$ (see Section 3.1). The cost of such a route $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_p, \sigma_{p+1})$ is given by

$$c(\sigma) = \sum_{i=0}^{p} c_{f_i, e_{i+1}} + \sum_{i=1}^{p} \hat{c}_{e_i, f_i}. \tag{1}$$

Next, we describe the elements of the Balas-Simonetti neighborhood for the CluVRP. Let the integer $k \geq 1$ be given and fixed. We define the Balas-Simonetti neighborhood of an CluVRP route $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_p, \sigma_{p+1})$ as all routes $\sigma' = (\sigma'_0, \sigma'_1, \ldots, \sigma'_p, \sigma'_{p+1})$ that fulfill the following conditions:

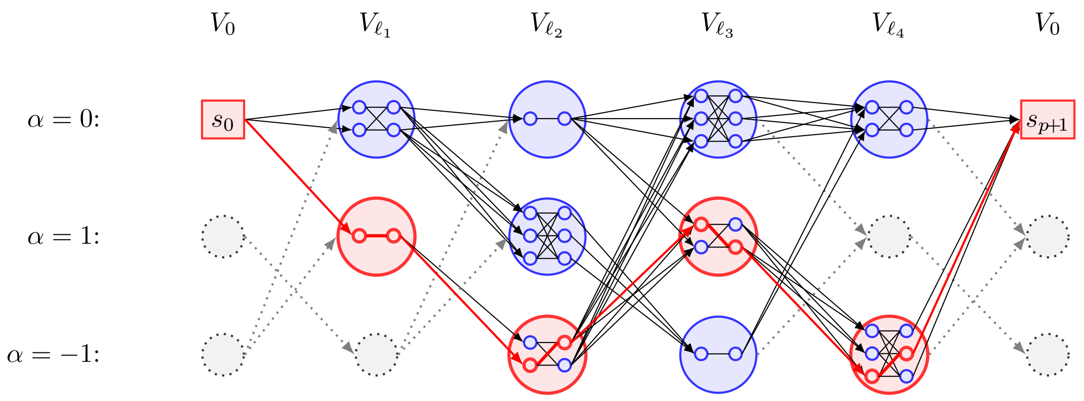

Figure 2: Auxiliary graph $\widehat{G}_k^*$ for $k = 2$, current solution $\sigma = (\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5)$ with $\sigma_0 = \sigma_5 = (0, \{0\}, 0)$ and $p = 4$ customer clusters visited in the sequence $V_{\ell_1}, V_{\ell_2}, V_{\ell_3}, V_{\ell_4}$. The neighbor $\sigma' = (\sigma'_0, \sigma'_1, \sigma'_2, \sigma'_3, \sigma'_4, \sigma'_5) \in \mathcal{N}_2^{BS}(\sigma)$ implied by the highlighted s_0-s_{p+1}-path visits the customer clusters in the sequence $V_{\ell_2}, V_{\ell_1}, V_{\ell_4}, V_{\ell_3}$.

(i) the ith triplet is $\sigma'_i = (e'_i, V_{\ell_{\pi(i)}}, f'_i)$ with arbitrary $e'_i, f'_i \in V_{\ell_{\pi(i)}}$, where

(ii) π is a permutation of $\{0, 1, \dots, p, p+1\}$ with $\pi(0) = \pi(p+1) = 0$, and if $g + k \leq h$ for a pair of indices $g, h \in \{1, 2, \dots, p\}$ then $\pi(g) \leq \pi(h)$ must hold.

In this case, we write $\sigma' \in \mathcal{N}_k^{BS}(\sigma)$. Note that we allow $k = 1$ here in contrast to the ATSP where $k \geq 2$ is required. For the CluVRP, $\mathcal{N}_1^{BS}(\sigma)$ does not at all permute the clusters but allows to arbitrarily modify the entry-exit combinations of all visited clusters. Such a neighborhood has been defined in the context of routing with service mode choice, e.g., the selection of the traversal directions in single- and multiple-vehicle arc-routing problems (Irnich, 2008; Bode and Irnich, 2012) and vehicle-routing problems with more general service mode choices (Vidal, 2016). Its has been shown there that optimal choices can be determined efficiently using DP techniques.

For the general case of $k \geq 1$, we show that an optimal combination of cluster permutation and all entry-exit nodes can be determined by solving again a source-to-sink shortest path problem in an auxiliary network $\widehat{G}_k^*$. We start by defining the structure of this auxiliary network. It has a macroscopic and a microscopic level, as visualized in Figure 2. The macroscopic level has *cluster nodes* that represent the depot and the p clusters. As the depot 0 is also the cluster V_0, we do not distinguish between depot and clusters in the following. The cluster nodes are permuted using the same auxiliary graph G_k^* as in the ATSP (see Section 3.1.1). Hence, the states s of G_k^* are copies of the clusters V_{ℓ_i} and the α-values allow us to refer to the associated original cluster. In Figure 2 with $k = 2$, the cluster nodes can only move one position backward or one position forward or stay at the same position.

At the microscopic level, each cluster V_{ℓ_i} is described by all possible triplets (e_i, V_{ℓ_i}, f_i) modeled by a complete bipartite graph. The first/left partition consists of the entry nodes $e_i \in V_{\ell_i}$, while the second/right partition consists of the exit nodes $f_i \in V_{\ell_i}$. Each edge in the bipartite graph refers to a specific triplet, and vice versa. Hence, the cost of an arc (e_i, f_i) is defined as the cost $\hat{c}_{e_i,f_i}$ of a Hamiltonian e_i-f_i-path. Note that in case of $e_i = f_i$ and $\lambda_i > 1$ this cost was defined as the large number M making choices with identical entry and exit unattractive for non-trivial clusters.

Finally, we have to define the cost of the arcs connecting different clusters. Connecting states s of stage i with states s' of stage $i + 1$ is simple. The arc connecting triplet (e_i, V_{q_i}, f_i) with triplet $(e_{i+1}, V_{r_{i+1}}, f_{i+1})$ receives the cost $c_{f_i,e_{i+1}}$. Now, each s_0-s_{p+1}-path in $\widehat{G}_k^*$ uniquely corresponds to a neighbor σ' of σ with cost $c(\sigma')$ as defined in (1).

In Figure 2, the given route σ starts at the depot cluster V_0, then visits the four clusters in the sequence $V_{\ell_1}, V_{\ell_2}, V_{\ell_3}, V_{\ell_4}$, and returns to the depot cluster V_0. The entry-exit combinations of σ are unimportant for describing its neighbors. The highlighted s_0-s_{p+1}-path is the neighbor solution σ' that visits the customer clusters in the sequence $V_{\ell_2}, V_{\ell_1}, V_{\ell_4}, V_{\ell_3}$. The first visited cluster V_{ℓ_2} contains only one customer so that entry and exit are identical to this customer. The second visited cluster V_{ℓ_1} is entered via its second and exited via its first customer, while for the third visited cluster V_{ℓ_4} it is reverse. The last visited cluster V_{ℓ_3} is entered via its third customer and exited via its second customer.

It is straightforward to generalize the complexity results known for the ATSP. Here, for a given CluVRP route $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_p, \sigma_{p+1})$, the DP algorithm to determine a best neighbor solution can be implemented requiring $\mathcal{O}\left(p\lambda_{\max}^2 k^2 2^k\right)$ time and space, where $\lambda_{\max} = \max_{1\leq i\leq p} \lambda_{\ell_i}$ is the size of the largest visited cluster. In particular, the search effort is linear in the number p of visited clusters and linear in the number of entry-exit combinations (which is bounded by $\lambda_{\max}^2$).

3.3. Cluster Neighborhoods and VND

In this section, we present a variant of VND (Hansen and Mladenović, 2001) that combines the single-route Balas-Simonetti neighborhood of the last section with several neighborhoods that can exchange clusters between routes. Since in the CluVRP customers of same clusters have to be visited contiguously, all neighborhoods move complete clusters. We can therefore re-use known neighborhoods from the CVRP by letting them operate on sequences of clusters. The CVRP neighborhoods that we adapt to the CluVRP are the (subsequence) relocation neighborhood *κ-Relocate*, the *Swap* neighborhood, and the *2-opt** neighborhood.

Before we describe these neighborhoods precisely, we formalize two strategies for determining new entry/exit combinations after the movement of clusters. In the *fixed version*, the entry/exit decisions are kept fixed as given by the current solution. The three neighborhoods can only alter the grouping of the clusters. In contrast, the *flex version* allows changing particular entry-exit combinations in the following ways:

Connect When two subroutes $\sigma^1 = (\ldots, \sigma^1_{i-1}, \sigma^1_i)$ with $\sigma^1_i = (e^1_i, V^1_{\ell_i}, f^1_i)$ and $\sigma^2 = (\sigma^2_j, \sigma^2_{j+1}, \ldots)$ with $\sigma^2_j = (e^2_j, V^2_{\ell_j}, f^2_j)$ are concatenated, the exit of cluster σ^1_i and the entry of cluster σ^2_j can be modified. We minimize the value $\hat{c}_{e^1_i, f'^1_i} + c_{f'^1_i, e'^2_j} + \hat{c}_{e'^2_j, f^2_j}$ over $(f'^1_i, e'^2_j) \in V^1_{\ell_i} \times V^2_{\ell_j}$. Note that e^1_i and f^2_j are still kept fixed. The resulting subroute is $(\ldots, \sigma^1_{i-1}, \sigma'^1_i, \sigma'^2_j, \sigma^2_{j+1}, \ldots)$ with $\sigma'^1_i = (e^1_i, V^1_{\ell_i}, f'^1_i)$ and $\sigma'^2_j = (e'^2_j, V^2_{\ell_j}, f^2_j)$.

Insert Inserting a cluster $\sigma_a = (e_a, V_{\ell_a}, f_a)$ into route $\sigma = (\ldots, \sigma_i, \sigma_j, \ldots)$ between σ_i and σ_j is done by minimizing $c_{f_i, e'_a} + \hat{c}_{e'_a, f'_a} + c_{f'_a, e_j}$ over $(e'_a, f'_a) \in V_{\ell_a} \times V_{\ell_a}$. Note that σ_i and σ_j are kept fixed. The resulting route is $\sigma' = (\ldots, \sigma_i, \sigma'_a, \sigma_j, \ldots)$ with $\sigma'_a = (e'_a, V_{\ell_a}, f'_a)$.

Note that the computational effort for minimization is in both cases bounded by $\mathcal{O}\left(\lambda_{\max}^2\right)$, where $\lambda_{\max}$ is the size of the largest cluster.

In the following, the relocation, swap, and 2-opt* neighborhoods are considered in both versions, fixed and flexible. For the brief description of the actual neighborhoods, we notice that no more than two routes are involved in any operation. We denote these two routes by $\sigma^1 = (\sigma^1_0, \sigma^1_1, \ldots, \sigma^1_{i-1}, \sigma^1_i, \sigma^1_{i+1}, \ldots, \sigma^1_p, \sigma^1_{p+1})$ and $\sigma^2 = (\sigma^2_0, \sigma^2_1, \ldots, \sigma^2_{j-1}, \sigma^2_j, \sigma^2_{j+1}, \ldots, \sigma^2_q, \sigma^2_{q+1})$.

Relocate Neighborhood. The neighborhood $\mathcal{N}^{\kappa\text{-reloc}}$ contains all CluVRP solutions that result from the removal of a subsequence of κ consecutive clusters from its current position and the insertion of the subsequence into another route or the same route at another position.

For $\kappa = 1$, the cluster σ^1_i is removed from σ^1 and inserted after σ^2_j into σ^2 resulting in the two new routes $\sigma'^1 = (\sigma^1_0, \sigma^1_1, \ldots, \sigma'^1_{i-1}, \sigma'^1_{i+1}, \ldots, \sigma^1_p, \sigma^1_{p+1})$ and $\sigma'^2 = (\sigma^2_0, \sigma^2_1, \ldots, \sigma^2_{j-1}, \sigma^2_j, \sigma'^1_i, \sigma^2_{j+1}, \ldots, \sigma^2_q, \sigma^2_{q+1})$. In the fixed version, all triplets remain unchanged so that $\sigma'^1_{i-1} = \sigma^1_{i-1}$, $\sigma'^1_{i+1} = \sigma^1_{i+1}$, and $\sigma'^1_i = \sigma^1_i$. In the flex version, the new routes are derived by applying *Connect* to the subroutes $(\sigma^1_0, \sigma^1_1, \ldots, \sigma^1_{i-1})$ and $(\sigma^1_{i+1}, \ldots, \sigma^1_p, \sigma^1_{p+1})$ to produce σ'^1 and by applying *Insert* to the subroutes $(\sigma^2_0, \sigma^2_1, \ldots, \sigma^2_{j-1}, \sigma^2_j)$ and $(\sigma^2_{j+1}, \ldots, \sigma^2_q, \sigma^2_{q+1})$ and the relocated cluster σ^1_i to produce σ'^2.

For $\kappa > 1$, the order of the κ inserted clusters can change, too (this generates a finite set of permutations, small whenever κ is small). We use *1-Relocate* and *2-Relocate*, which are considered two different neighborhoods in the following.

Swap Neighborhood. The neighborhood $\mathcal{N}^{\text{swap}}$ contains all CluVRP solutions that result from the swapping of two clusters either from the same or from two different routes. In the latter case, swapping σ_i^1 and σ_j^2 gives the two new routes $\sigma'^1 = (\sigma_0^1, \sigma_1^1, \ldots, \sigma_{i-1}^1, \sigma_j'^2, \sigma_{i+1}^1, \ldots, \sigma_p^1, \sigma_{p+1}^1)$ and $\sigma'^2 = (\sigma_0^2, \sigma_1^2, \ldots, \sigma_{j-1}^2, \sigma_i'^1, \sigma_{j+1}^2, \ldots, \sigma_q^2, \sigma_{q+1}^2)$. Depending on the version fixed or flex, either all triplets remain unchanged or *Insert* is applied for deriving both σ'^1 and σ'^2.

2-Opt Neighborhood.* The neighborhood $\mathcal{N}^{\text{2-opt*}}$ comprises all CluVRP solutions that result from cutting two different routes into front part and back part and concatenating each front with the other back part. Cutting after σ_i^1 and σ_j^2, respectively, produces two new routes $\sigma'^1 = (\sigma_0^1, \ldots, \sigma_{i-1}^1, \sigma_i'^1, \sigma_{j+1}'^2, \sigma_{j+2}^2 \ldots, \sigma_q^2, \sigma_{q+1}^2)$ and $\sigma'^2 = (\sigma_0^2, \ldots, \sigma_{j-1}^2, \sigma_j'^2, \sigma_{i+1}'^1, \sigma_{i+2}^1 \ldots, \sigma_p^1, \sigma_{p+1}^1)$. In the fixed version, all triplets remain unchanged. In the flex version, the reconnection is done with the procedure *Connect*.

Size and Search Complexity. The size of the neighborhoods $\mathcal{N}^{\kappa\text{-reloc}}$ (for $\kappa = 1$ and 2), $\mathcal{N}^{\text{swap}}$, and $\mathcal{N}^{\text{2-opt*}}$ increases quadratically with the overall number N of the clusters. Therefore, the effort to search them is in the fixed version bounded by $\mathcal{O}(N^2)$. Since the effort for the procedure *Connect* and *Insert* is bounded by $\mathcal{O}(\lambda_{\max}^2)$, the overall search effort is limited by $\mathcal{O}(N^2\lambda_{\max}^2)$ in the flex version.

Neigborhood	fixed version (no entry-exit modification)	flexible version (with entry-exit modification)
Balas-Simonetti $\mathcal{N}_{k_{\text{VND}}}^{BS}$		1, best improvement
1-Relocate $\mathcal{N}^{\text{1-reloc}}$	2, first improvement	5, first improvement
2-Opt* $\mathcal{N}^{\text{2-opt}}$	3, first improvement	6, first improvement
2-Relocate $\mathcal{N}^{\text{2-reloc}}$	4, first improvement	7, first improvement
Swap $\mathcal{N}^{\text{swap}}$	4, first improvement	7, first improvement

Table 1: Priorities and pivoting strategy of the nine VND neighborhoods

Our version of VND uses the nine neighborhoods listed in Table 1. In pre-tests we also analyzed different possible sequences of applying the neighborhoods and different pivoting strategies. Concerning the sequence of neighborhoods, it is common practice to apply those neighborhoods first that can be searched quickly. Therefore, we start with the linear (in the route length) neighborhood $\mathcal{N}_{k_{\text{VND}}}^{BS}$ (the choice of a reasonable parameter k_{VND} is analyzed in detail in Section 4.2), then apply all four neighborhoods in the fixed version before those using the flex version. Pre-tests also revealed that $\mathcal{N}^{\text{1-reloc}}$ should be favored over $\mathcal{N}^{\text{2-opt}}$. In turn, $\mathcal{N}^{\text{2-opt}}$ should be favored over $\mathcal{N}^{\text{2-reloc}}$ and $\mathcal{N}^{\text{swap}}$. Moreover, we found that it is advantageous to alternate between the two latter neighborhoods $\mathcal{N}^{\text{2-reloc}}$ and $\mathcal{N}^{\text{swap}}$. Finally, a first improvement pivoting strategy was most of the time faster without deteriorating the solution quality for the cluster exchange neighborhoods (note that $\mathcal{N}_{k_{\text{VND}}}^{BS}$ is always searched with a best improvement strategy due to the DP algorithm using the auxiliary network $\widehat{G_k^*}$). Based on these observations, the final design of the VND is summarized in Algorithm 1. The priorities and pivoting strategies of all nine neighborhoods are given in Table 1.

There are two more findings that helped us to significantly accelerate the VND approach. It is not necessary to apply the clustered version of the Balas-Simonetti neighborhood to input solutions of the VND. Therefore, *prio* is initialized to 1 (see Step 2) so that the first searched neighborhood is the fixed version of $\mathcal{N}^{\text{1-reloc}}$ (cf. Table 1). Second, for reasonably (small) parameters k_{VND}, the neighborhood $\mathcal{N}_{k_{\text{VND}}}^{BS}$ can be searched quickly. However, we found that almost always a solution once improved with $\mathcal{N}_{k_{\text{VND}}}^{BS}$ cannot be improved with the same neighborhood directly afterwards. It means that no proper local search with $\mathcal{N}_{k_{\text{VND}}}^{BS}$ is necessary. We therefore apply $\mathcal{N}_{k_{\text{VND}}}^{BS}$ only once (deviating from Algorithm 1) and directly continue with the fixed version of $\mathcal{N}^{\text{1-reloc}}$.

Algorithm 1: VND with neighborhood priorities and different pivoting strategies

```
   Input: Initial solution $x = (\sigma^1, \sigma^2, \ldots, \sigma^m)$,
          Set of neighborhoods $\{\mathcal{N}\}$ with priorities $prio(\mathcal{N})$ and pivoting strategy $pivot(\mathcal{N})$
1  $iter := 0$
2  $prio := 1$
3  repeat
4      $prio := prio + 1$
5      $nb :=$ number of neighborhoods $\mathcal{N}$ with $prio(\mathcal{N}) = prio$
6      repeat
7          $i := iter$ modulo $nb$
8          $\mathcal{N} :=$ the $i$th neighborhood with $prio(\mathcal{N}) = prio$
9          Search $\mathcal{N}$ with pivoting strategy $pivot(\mathcal{N})$ for improving neighbors
10         if improving neighbor $x' \in \mathcal{N}(x)$ found then
11             $x := x'$
12             $prio := 0$
13     until ($prio = 0$) or (up to $nb$ times)
14 until $prio > \max_{\mathcal{N}} prio(\mathcal{N})$
   Output: Local optimum $x = (\sigma^1, \sigma^2, \ldots, \sigma^m)$ w.r.t. all neighborhoods $\{\mathcal{N}\}$
```

3.4. LNS Operators

Large neighborhood search (LNS) was originally introduced by Shaw (1998) for the CVRP. A similar idea, there called *ruin and recreate*, can be found in (Schrimpf *et al.*, 2000). Pisinger and Ropke (2010) give an overview of different LNS approaches and extensions.

The basic approach starts from a given feasible starting solution and repeats destroy and repair steps until a stopping criterion lets the LNS terminate. Parts of the current solution are destroyed by a *destroy operator*. For VRPs, this destroy operator is typically the removal of a subset of the customers from their routes. The resulting partial solution is then restored again by a *repair operator*, which is (in VRPs) the reinsertion of the removed customers into the same or other routes at possibly different positions.

Both destroy and repair operators often include some randomness. For example, the customer subset including the decision of its size can vary from one iteration to the next. While Shaw (1998) suggests to increase the size if no improvement is found for a certain number of iterations, Ropke and Pisinger (2006a) always choose the size randomly out of a given range. The new solution is accepted as the current solution depending on an acceptance criterion. Moreover, LNS keeps track of the best found solution.

Different acceptance criteria have been used. While Shaw (1998) only accept improving solutions, Ropke and Pisinger (2006a,b) use simulated annealing's Metropolis acceptance criterion. For the pickup and delivery problem with time windows, Ropke and Pisinger coined the idea of an *adaptive* LNS (ALNS): Instead of using only one removal and one repair operator, they use several operators for removal and repair. Operators are then randomly selected on the basis of weights, which are updated depending on the success of their corresponding operator in previous iterations.

We describe our LNS as a large multiple neighborhood search (LMNS, Pisinger and Ropke, 2007) because we use several destroy and repair operators but their weights are kept fixed over the LNS iterations. Since the detailed analysis in Section 4.1.2 shows that our LMNS is not very sensitive to the modification of weights, we decided for a simple design without an adaptive weights modification component (in contrast to ALNS). Specific for our LMNS is also that we improve solutions after the repair step with the help of the VND described in Section 3.3. Such a post-optimization of solutions with the help of a local search was also used by Ropke (2009).

Next, we describe the destroy and repair operators in Sections 3.4.1 and 3.4.2 and provide a summary of the overall algorithm in Section 3.5.

3.4.1. Destroy Operators

As the cluster neighborhoods of Section 3.3 do, our destroy and repair operators only remove and insert entire clusters instead of individual customers. After removing a cluster, the exit of the preceding and the entry of the succeeding cluster are connected without modifying the current entry-exit combinations.

We use the following four different destroy operators:

1. *Random destroy* removes τN clusters at random (Ropke and Pisinger, 2006a), where the parameter τ controls the percentage of the clusters to be removed.
2. *Related destroy* is a variant of the destroy method originally proposed by Shaw (1998). At the beginning, an initial cluster is randomly chosen and removed. Afterwards $\tau N - 1$ clusters closest to the initial cluster are also removed. Again, the parameter τ controls the percentage of clusters to be removed. We compute the distance between two clusters V_g and V_h as $\min_{(i,j)\in V_g \times V_h} c_{ij}$.
3. *Worst destroy* is described in detail by (Ropke and Pisinger, 2006a) and works as follows: For every cluster, we calculate the improvement that would occur if the cluster was removed from the current solution. All clusters are sorted by decreasing improvements in the list L. For τN iterations, the cluster at position $pos = y^{\rho}\,|L|$ is removed from L, where $y \in [0, 1)$ is a uniformly distributed random number. Also here, the parameter τ describes the percentage of clusters to be removed. The additional parameter $\rho \geq 1$ controls the degree of randomization: The larger the value of ρ, the more likely the operator chooses clusters at the front of list L, i.e., clusters with a high cost improvement when removed. Improvement values and the sorted list L are updated in every iteration.
4. *Route destroy* picks a route at random and removes it.

3.4.2. Repair Operators

To reinsert the removed clusters we implemented the following two repair operators:

1. *Nearest repair* reinserts all removed clusters according to their distance to the partial solution. Depending on the insertion costs the nearest cluster is inserted before or after the closest cluster of the partial solution. If the closest cluster is the depot, a new route is generated. However, the overall number of routes is bounded by m.
2. By *Best repair* clusters are reinserted using a largest-demand-first rule. Insertion costs are calculated for every feasible position (using procedure *Insert*) and the current cluster is inserted at its best position. If the number of routes was reduced by the destroy operator, clusters with largest demand are used to restore the required number of m routes.

Both operators use the procedure *Insert* as described in Section 3.3 to execute the move.

3.5. Overall LMNS Algorithm

The pseudo-code of the overall LMNS approach is shown in Algorithm 2. It combines all components presented in the previous sections. We briefly summarize the steps.

In Step 1, the preprocessing determines the intra-cluster routes for each pair of entry and exit (Section 3.1). A starting solution is computed in Step 2 with a *regret-based savings algorithm* tailored to the CluVRP. A *savings value* is calculated for each pair (V_g, V_h) of clusters as

$$sav_{g,h} = c(\sigma_0, \sigma_g, \sigma_0) + c(\sigma_0, \sigma_h, \sigma_0) - c(\sigma_0, \sigma_g, \sigma_h, \sigma_0),$$

where $\sigma_0 = (0, V_0, 0)$ is the depot cluster/triplet, and $\sigma_g = (e_g, V_g, f_g)$ and $\sigma_h = (e_h, V_h, f_h)$ are the gth and hth cluster/triplet. The savings value depends on the choice of entry and exit points e_g, e_h, f_g, and f_h, and we determine cost-minimizing combinations in $\mathcal{O}\left(\max\{\lambda_g, \lambda_h\}^2\right)$ time by solving a small DP over the

Algorithm 2: LMNS algorithm

```
   Input: Iterations $It_{\text{ATSP}}$ and $It_{\text{LMNS}}$
          Parameters $k_{\text{ATSP}}$, $k_{\text{VND}}$, and $k_{\text{LMNS}}$ of Balas-Simonetti neighborhoods
          Weights $(\psi^{random}, \psi^{related}, \psi^{worst}, \psi^{route})$ and $(\omega^{nearest}, \omega^{best})$ of removal and repair
          operators
          Parameters $\epsilon$, $\tau_{\min}, \tau_{\max}$, and $\rho$
1  Preprocessing($It_{\text{ATSP}}$, $k_{\text{ATSP}}$)
2  $x := x^{accepted} := x^{best} :=$ Regret-based Savings Algorithm()
3  for $iter := 1, \ldots, It_{\text{LMNS}}$ do
4      if $x$ is feasible then
5          $x := \text{VND}(k_{\text{VND}}, x)$
6          $x :=$ Single Improvement with $\mathcal{N}^{BS}_{k_{\text{LMNS}}}(x)$
7          if $c(x) < c(x^{best})$ then
8              $x^{best} := x$
9          if AcceptanceCriterion($\epsilon$, $x$, $x^{best}$) 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 $\text{Op}^{repair}$ according to weights $(\omega^{nearest}, \omega^{best})$
14     $x := \text{Op}^{repair}(\text{Op}^{destroy}(\tau, \rho, x^{accepted}))$
```

auxiliary network $\widehat{G^*_k}$ for $k = 1$. In contrast to the classical savings algorithm, we calculate a *regret value* for each cluster V_g as the difference between its best and second best possible saving, i.e.,

$$\text{regret}(g) := \left(\max_h sav_{g,h} \right) - \left(\max_h{}^{(2)} sav_{g,h} \right),$$

where $\max^{(2)}$ denotes the second largest (possibly identical) value among all feasible savings. As in the classical savings algorithm, a saving becomes infeasible if either V_g and V_h are already inserted into the same route or, if in different routes, the demand associated with their routes exceeds the vehicle capacity Q. The largest regret value regret(g) determines the cluster V_g to be inserted first. Regret values are updated in every iteration of the savings algorithm.

The main loop of the LMNS comprises the Steps 3–14 and is repeated for It_{LMNS} iterations. Infeasible solutions x can result from combined destroy and repair operations performed in Step 14. However, we accept only feasible solutions as *accepted solutions* x^{accept}. In Step 5, feasible solutions are always post-optimized with the VND algorithm described in Section 3.3. Afterwards, in Step 6, each route σ^r for $r \in \{1, 2, \ldots, m\}$ of the current solution $x = (\sigma^1, \sigma^2, \ldots, \sigma^m)$ is post-optimized with the Balas-Simonetti neighborhood $\mathcal{N}^{BS}_{k_{\text{LMNS}}}$. Hence, our clustered version of the Balas-Simonetti neighborhood is applied at two different places in the LMNS approach, i.e., inside the VND and in a post-optimization step. Note that the two parameters k_{VND} and k_{LMNS} can differ, and we present a detailed parameter study in Section 4.2 for finding a reasonable pair $(k_{\text{VND}}, k_{\text{LMNS}})$ providing a good computation time to quality tradeoff.

In Steps 7 and 8, the best solution found is updated when necessary. Depending on the acceptance criterion, the accepted solution is also updated in Steps 9 and 10. Our LMNS acceptance criterion is based on the record-to-record principle. The current solution x is accepted if $c(x) < (1 + \epsilon)\, c(x^{best})$.

Finally, the percentage of clusters to destroy and the specific destroy and repair operators for the current LMNS iteration are randomly chosen in Steps 11–13. The current solution is then in Step 14 destroyed and repaired with the six operators discussed in Section 3.4, creating the starting solution for the next iteration.

4. Computational Results

All computations are performed on a standard PC equipped with MS Windows 7 running on an Intel(R) Core(TM) i7-5930K CPU clocked at 3.5 GHz and with 64 GB RAM of main memory. All algorithms were coded in C++ and compiled with MS Visual Studio 2010 in release mode.

We test our LMNS algorithm on three different benchmark sets that were also used in previous studies in the literature. All CluVRP benchmarks were derived from CVRP benchmarks using the cluster generator described in detail in (Bektaş *et al.*, 2011) and (Fischetti *et al.*, 1997). The cluster generator uses a parameter θ to specify the desired average number of customers per cluster. Then $N = \lceil (n+1)/\theta \rceil$ customer clusters are built.

The first instance set `GVRP` by Bektaş *et al.* (2011) comprises the ten smallest CluVRP instances with 100 to 262 nodes and $\theta = 2$ or 3. They are available online at `http://www.personal.soton.ac.uk/tb12v07/gvrp.html`. The second instance set `Golden` was proposed by Battarra *et al.* (2014) and is based on the well-known CVRP instances by Golden *et al.* (1998). It contains eleven groups, each with 20 instances, all based on identical customer sets denoted by `Golden1` to `Golden20` but differing in the value of θ ranging from 5 to 15. The number of nodes in these instances varies between 201 and 484. The third set `Li` consists of twelve large-scale instances with 561 to 1 201 nodes. It is based on the CVRP instances of Li *et al.* (2005). Vidal *et al.* (2015) generated them using a value of $\theta = 5$. In all three benchmark sets, the number of vehicles m is given for each instance. It is not allowed to use less vehicles and our algorithm enforces that each vehicle serves at least one cluster.

For each instance, LMNS is run ten times each with a different random seed. The solution quality is measured by the *gap* (in percent) between the solution value z and the best known solution BKS. It is calculated as $100\,(z - \mathrm{BKS})/\mathrm{BKS}$. In addition, *Gap Avg.* is the average gap per instance over ten runs, while *Gap Best* is the smallest gap obtained over the ten runs. All *computation times T* are given in seconds.

4.1. Parameter Study

In the first series of experiments, we determine reasonable values for the parameters of our LMNS metaheuristic. In both the preprocessing and the actual LMNS, we have to find a good tradeoff between required computation time and solution quality.

4.1.1. Parameters for Preprocessing

Quality of the preprocessing is crucial because no later step of the LMNS algorithm can revise a possibly incorrect SHP solution. Hence, we must carefully assess the quality of the preprocessing, which is however straightforward because Vidal *et al.* (2015) provide exact solutions to all SHPs.

We systematically try different combinations of λ_{BS}, k_{ATSP}, and It_{ATSP}. The most important observations are the following: The limited DP approach for exactly solving small-sized SHPs is only sufficiently fast when clusters V_h have no more than ten customers. We therefore set $\lambda_{\mathrm{BS}} = 10$. In the combined ILS/VND for solving ATSPs, we must also calibrate the Balas-Simonetti neighborhood parameter k_{ATSP} and the number It_{ATSP} of ILS iterations. Clearly, the parameter k_{ATSP} must be chosen much smaller than $\lambda_{\mathrm{BS}} = 10$, since multiple VND iterations search over the Balas-Simonetti neighborhood. After testing several combinations we can state that a good compromise with respect to both time consumption and solution quality is the combination $k_{\mathrm{ATSP}} = 3$ and $It_{\mathrm{ATSP}} = 50$.

We summarize what criteria we studied to find the combination $k_{\mathrm{ATSP}} = 3$ and $It_{\mathrm{ATSP}} = 50$. Over all instances, the preprocessing must consider 1 074 423 entry-exit combinations coming from 11 468 clusters. Among these, the ILS metaheuristic considers 827 814 SHPs for entry-exit combinations imposed by 2 483 clusters with eleven or more customers. ILS fails to find an optimum in 2.6 % of the cases, i.e., in 21 478 SHPs. Non-optimal solutions occur for 11.7 % of the clusters, where the smallest cluster contains 14 customers. In these more difficult clusters, on average 3.8 % of the SHPs are not solved to optimality. However, for 18 clusters and 118 SHPs, we identify better SHP solutions than the ones written into the instance files kindly sent to us by Battarra (2015).

The overall quality of our preprocessing could clearly be increased by choosing larger values for $k_{\mathrm{ATSP}} = 3$ and $It_{\mathrm{ATSP}} = 50$. However, the subsequent experiments (a posteriori) confirm the above decision: A suboptimal SHP is chosen only once in the best CluVRP solutions (computed in Section 4.3). However, all entry-exit combinations are correct. If instead the exact SHP solution of Battarra *et al.* (2014) were chosen, the improvement is one unit of cost. It is certainly much more effective to invest additional time into the actual LMNS instead of intensifying the preprocessing.

4.1.2. Parameter for LMNS

The LMNS metaheuristic uses several parameters that have to be defined. To find a good parameter set, we follow a strategy similar to the one used by Ropke and Pisinger (2006a). We start with a basic setting found during pre-tests. Pre-tests have revealed that for the parameters $(\epsilon, \tau_{\min}, \tau_{\max}, \rho)$ the values $(0.005, 10, 40, 10)$ make sense, i.e., the record-to-record acceptance criterion uses the factor $(1+\epsilon) = 1.005$ to compare with the currently best found solution, between $\tau_{\min} = 10$ and $\tau_{\max} = 40$ percentage of the clusters are destroyed, and the randomization exponent ρ is chosen as 10 in the worst removal operator.

First, we determine the (non-adaptive) weights for the destroy and repair operators. Here, we find that the chosen setup with four different destroy and two repair operators is not very sensitive with respect to the choice of the weights. Therefore, we apply the two repair operators with identical probabilities $(\omega^{nearest}, \omega^{best}) = (0.5, 0.5)$. For the four destroy operators, the only important finding is that the route removal operator does not need to be applied as often as the other three operators. Hence, we chose $(\psi^{random}, \psi^{related}, \psi^{worst}, \psi^{route}) = (0.3, 0.3, 0.3, 0.1)$ for the weights.

Second, we test the usefulness of each and every operator: We find that using only one destroy and one repair operator (eight possible setups) is clearly outperformed by the combination of all operators. Moreover, for each single operator we test whether it is redundant. Here, we keep all other five operators and their weights in the same ratio as given above. For example, without the random removal operator, the other removal operators receive weights $(\psi^{related}, \psi^{worst}, \psi^{route}) = (0.3, 0.3, 0.1)/0.7$.

	w/o destroy operator				w/o repair operator		*All*
	Random	*Related*	*Worst*	*Route*	*Nearest*	*Best*	*Operators*
Time T	46	44	42	45	40	49	45
Gap Best [%]	0.032	0.032	0.035	0.035	0.039	0.047	0.025
# BKS	213	208	214	213	205	201	217

Table 2: Comparison of LMNS using different destroy and repair operators

Accordingly, Table 2 shows the results of disabling a single operator in comparison to the version (called *All Operators*) that uses all six operators. These tests are performed with $It_{\text{LMNS}} = 5\,000$ and the combination $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ (see the next section for a study on reasonable $(k_{\text{VND}}, k_{\text{LMNS}})$ combinations). Computation times T are very similar in all seven settings (between 42 and 49 seconds per CluVRP instance on average). Using all operators leads to a LMNS that computes 217 best known solutions and at the same time the lowest overall *Gap Best* of 0.025 %. The six other variants compute between 201 and 214 *best known solutions* (BKS) with an average gap of at least 0.032 %. This is a clear indication that all operators contribute to the quality of LMNS. Hence, we use all six operators with the weights given above for the remaining experiments.

4.2. Usefulness of the Balas-Simonetti Neighborhood

In this section, we analyze the generalized version of the Balas-Simonetti neighborhood (see Section 3.2) that is a fundamental component of our LMNS metaheuristic. Recall that it is applied at two different places, i.e., inside the VND as one of the neighborhoods and as a post-optimization procedure. The corresponding pair of parameters is $(k_{\text{VND}}, k_{\text{LMNS}})$. We set $k_{\text{VND}} = 0$ and $k_{\text{LMNS}} = 0$, respectively, to indicate that the Balas-Simonetti neighborhood is not used in the VND and/or for post-optimization.

We test combinations $(k_{\text{VND}}, k_{\text{LMNS}}) \in \{0, 1, 3, 5, 7\} \times \{0, 1, 3, 5, 7\}$. Each so defined LMNS metaheuristic is run for $It_{\text{LMNS}} = 5\,000$ iterations. Figure 3 shows the results for all 242 CluVRP instances, comparing the average computation time T and the gap (best out of ten runs). All Pareto-optimal combinations are

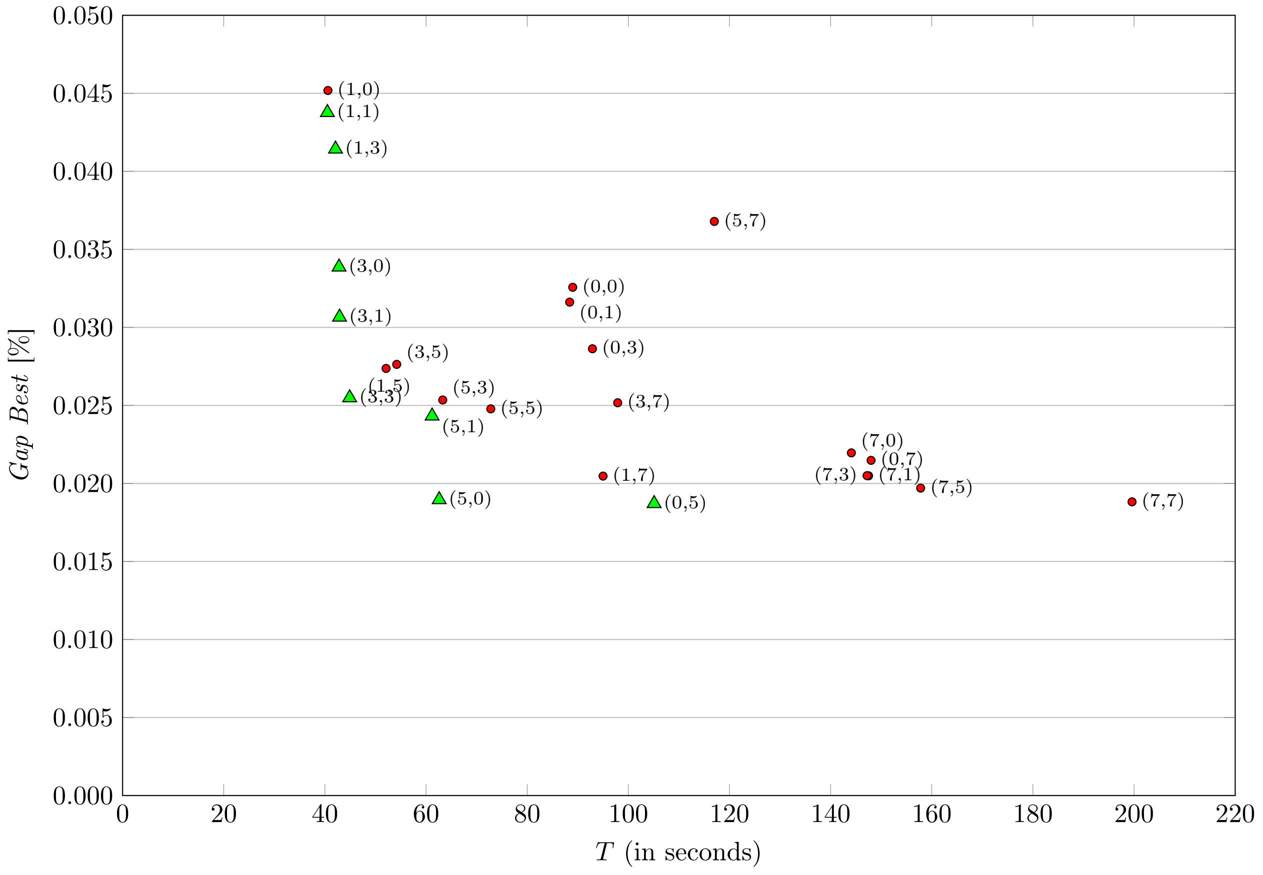

Figure 3: Comparison of LMNS with different combinations $(k_{\text{VND}}, k_{\text{LMNS}})$. Green triangles $\triangle$ indicate Pareto-optimal combinations, dominated combinations are indicated as red circles $\circ$.

marked by green triangles, all other by red circles. The combination $(0, 0)$ that does not at all make use of the Balas-Simonetti neighborhood in the LMNS iterations is clearly outperformed by many other configurations. In general, increasing k_{VND} and/or k_{LMNS} tends to improve the average solution quality at the cost of longer average computation times. However, there are also counterexamples such as the two combinations $(0, 7)$ and $(5, 7)$ where a larger neighborhood leads to an inferior solution quality and smaller computation times. The combination $(5, 7)$ is clearly an outlier, since a rather weak solution with a gap of 2.89 % for one instance of the `GVRP` benchmark is computed. When comparing combinations with small values $k_{\text{VND}}, k_{\text{LMNS}} = 0, 1$, and 3, the computational effort increases only very moderately with k_{LMNS} tending to produce significantly better results.

Overall, the two Pareto-optimal combinations $(3, 3)$ and $(5, 0)$ balance computation time and solution quality very well. We use both settings with $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ and $(5, 0)$ in the remainder.

4.3. Comparison to Result of Vidal et al. *(2015)*

This section provides a comparison of the LMNS and the UHGS approach by Vidal *et al.* (2015). When analyzing Tables 2 and 4 and Tables A2–A4 from the article Vidal *et al.* (2015), we detected some inconsistencies in the printed values. We informed the authors and Battarra and Vidal (2017) kindly confirmed that some entries in their tables are swapped/shifted between rows. An erratum is in preparation. In the following, we compare with the true values computed with the UHGS, sent to us by Battarra and Vidal (2017).

We run our LMNS using both settings $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ and $(k_{\text{VND}}, k_{\text{LMNS}}) = (5, 0)$ for $It_{\text{LMNS}} =$ 5 000 and 50 000 iterations, respectively. In addition, setting $(3, 3)$ is tested with 100 000 iterations and setting $(5, 0)$ with 75 000 iterations, finally leading to computation times comparable to what was reported for UHGS. For simplicity, the LMNS settings are denoted by $\text{LMNS}^{k_{\text{VND}},k_{\text{LMNS}}}_{It_{\text{LMNS}}}$ in the following, e.g., $\text{LMNS}^{3,3}_{5\,000}$ if LMNS with setting $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ is run for $It_{\text{LMNS}} = 5\,000$ iterations. Tables 3 and 4 summarize aggregated results grouped by setting and instance set. Herein, T is the total computation time in seconds and T_p the time spent with preprocessing, which does not depend on the setting.

	LMNS													UHGS				
		$It_{\text{LMNS}} = 5\,000$				$It_{\text{LMNS}} = 50\,000$				$It_{\text{LMNS}} = 100\,000$				(Vidal *et al.*, 2015)				
Set	T_p	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T_p	T	*Gap Best*	*Gap Avg.*	# BKS
`GVRP`	0.1	11	0.08	0.57	7	109	0.05	0.22	8	218	0.03	0.16	9	9.4	61	0.08	0.22	8
`Golden`	8.2	35	0.01	0.05	209	270	0.01	0.02	214	533	0.01	0.02	214	802.4	856	0.01	0.03	213
`Li`	5.3	264	0.20	0.40	1	2508	0.06	0.20	2	5072	0.04	0.16	3	314.7	660	0.02	0.16	10
Total	7.7	45	0.03	0.09	217	374	0.01	0.04	224	745	0.01	0.03	226	745.5	814	0.01	0.04	231

Table 3: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ and all benchmark sets (10 `GVRP` instances with the number n of customers ranging from 100 to 261 and an average number θ of nodes per cluster between 2 and 3, 220 `Golden` instances with n between 200 to 483 and θ between 5 and 15, and 12 `Li` instances with n between 560 to 1200 and $\theta = 5$).

	LMNS													UHGS				
		$It_{\text{LMNS}} = 5\,000$				$It_{\text{LMNS}} = 50\,000$				$It_{\text{LMNS}} = 75\,000$				(Vidal *et al.*, 2015)				
Set	T_p	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T_p	T	*Gap Best*	*Gap Avg.*	# BKS
`GVRP`	0.1	17	0.06	0.55	9	157	0.03	0.25	9	235	0.03	0.14	9	9.4	61	0.08	0.22	8
`Golden`	8.2	49	0.01	0.04	208	410	0.01	0.02	214	611	0.01	0.02	214	802.4	856	0.01	0.03	213
`Li`	5.3	347	0.13	0.37	1	3137	0.06	0.19	4	4836	0.05	0.17	4	314.7	660	0.02	0.16	10
Total	7.7	63	0.02	0.07	218	535	0.01	0.04	227	805	0.01	0.03	227	745.5	814	0.01	0.04	231

Table 4: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (5, 0)$ and all benchmark sets.

Comparing computation times, $\text{LMNS}^{3,3}_{5\,000}$ is faster than $\text{LMNS}^{5,0}_{5\,000}$ (45 vs. 63 seconds on average), but results in larger gaps (e.g. 0.03 % vs. 0.02 % for 'best of 10 runs') and finds one BKS less. Increasing the number of iterations It_{LMNS} in both cases reduces the gap to 0.01 % (*Gap Best*) and 0.03 % (*Gap Avg.*) but increases average computation times to 745 and 805 seconds, respectively. Over the test set comprising 242 instances, 226 BKS are found by $\text{LMNS}^{3,3}_{100\,000}$ and 227 BKS by $\text{LMNS}^{5,0}_{75\,000}$, which is less than the 231 BKS found by UHGS. However, the average gap produced by UHGS is worse (0.04 %) and its computational effort is higher (814 seconds on average). While our better bounds result from innovative LMNS components such as the generalized Balas-Simonetti neighborhood (see previous section), the key factor leading to the reduced computation times is our heuristic preprocessing.

We now analyze the instance sets separately: the small-sized `GVRP` instances are solved by $\text{LMNS}^{5,0}_{5\,000}$ with *Gap Best* = 0.06 % in 17 seconds, which compares favorably to UHGS with a gap of 0.08 % running for 61 seconds on average. Furthermore, we find nine of ten BKS including one new BKS, while UHGS finds eight. Compared to $\text{LMNS}^{5,0}_{5\,000}$, the setting $\text{LMNS}^{3,3}_{5\,000}$ performs slightly worse. In both cases, the average and best gap of LMNS can be reduced to values below those of UHGS (0.22 % and 0.08 %) by increasing the number of iterations: For example, $\text{LMNS}^{5,0}_{75\,000}$ gives an average gap of 0.14 % and *Gap Best* is reduced down to 0.03 %. However, our average computation times are then larger than those of UHGS.

Considering the `Golden` instances, LMNS clearly outperforms UHGS. The same *Gap Best* (0.01 %) is achieved in significantly shorter computation time, e.g., 35 seconds for $\text{LMNS}^{3,3}_{5\,000}$ compared to 856 seconds for UHGS. $\text{LMNS}^{3,3}_{5\,000}$ finds one BKS more than $\text{LMNS}^{5,0}_{5\,000}$, but produces larger average gaps (0.05 % and 0.04 % compared to 0.03 %). An increased number of iterations leads to 214 BKS and *Gap Avg.* = 0.02 %, independent from the LMNS settings, which is slightly better than 213 BKS and an average gap of 0.03 % for UHGS. LMNS times remain below those of UHGS on average.

For the `Li` instances, $\text{LMNS}^{3,3}_{5\,000}$ consumes 264 seconds and $\text{LMNS}^{5,0}_{5\,000}$ 347 seconds on average, which is faster than UHGS (660 seconds), but our gaps and the number of BKS found by LMNS are inferior. All LMNS gaps can be improved by increasing the number of iterations. However, we do not reach the excellent *Gap Best* of 0.02 % of UHGS, even with $\text{LMNS}^{3,3}_{100\,000}$ where the computational effort is high. On the positive side, both LMNS settings generate one new BKS for the `Li` benchmark. In addition, one further new BKS is found during experimentation with another setting (see detailed results in the Appendix).

	LMNS														UHGS				
		$It_{\text{LMNS}} = 5\,000$					$It_{\text{LMNS}} = 50\,000$				$It_{\text{LMNS}} = 100\,000$				(Vidal *et al.*, 2015)				
θ	T_p	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T_p	T	*Gap Best*	*Gap Avg.*	# BKS
5	0.5	47	0.04	0.10	16	463	0.00	0.04	19	926	0.00	0.04	19	66.1	158	0.03	0.08	16
6	1.2	39	0.06	0.12	17	371	0.05	0.08	19	744	0.05	0.07	19	94.3	168	0.06	0.11	17
7	1.6	33	0.00	0.06	20	317	0.00	0.00	20	636	0.00	0.00	20	113.6	183	0.00	0.04	20
8	3.0	32	0.00	0.02	19	288	0.00	0.01	19	577	0.00	0.01	19	204.7	259	0.00	0.02	20
9	4.4	30	0.00	0.01	20	264	0.00	0.00	20	525	0.00	0.00	20	264.1	315	0.00	0.01	20
10	6.0	30	0.03	0.06	19	240	0.03	0.04	19	478	0.03	0.03	19	511.0	561	0.00	0.02	20
11	7.6	29	0.01	0.03	19	221	0.01	0.03	19	436	0.01	0.03	19	357.7	403	0.00	0.01	20
12	9.8	30	0.01	0.04	19	212	0.01	0.01	19	415	0.01	0.01	19	974.2	1017	0.00	0.01	20
13	13.5	32	0.00	0.06	20	199	0.00	0.05	20	387	0.00	0.05	20	867.1	907	0.00	0.00	20
14	18.2	36	0.00	0.02	20	197	0.00	0.01	20	378	0.00	0.01	20	2283.6	2321	0.00	0.00	20
15	24.4	41	0.00	0.00	20	194	0.00	0.00	20	366	0.00	0.00	20	3090.4	3127	0.00	0.00	20
Total	8.2	35	0.01	0.05	209	270	0.01	0.02	214	533	0.01	0.02	214	802.4	856	0.01	0.03	213

Table 5: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ and the `Golden` instances grouped by average cluster size; each group comprises 20 instances.

The `Golden` instances are grouped by the average cluster size θ (ranging from five to 15) into eleven groups with 20 instances each. We present detailed results for each group in Tables 5 and 6. Here, the computation times for the preprocessing T_p are strongly increasing with the average size of the clusters. In turn, the number N of clusters decreases and this reduces the actual LMNS computation time. For the small number of 5 000 iterations, both effects almost balance the overall computation time T over different

		LMNS												UHGS				
		$It_{\text{LMNS}} = 5\,000$				$It_{\text{LMNS}} = 50\,000$				$It_{\text{LMNS}} = 75\,000$				(Vidal *et al.*, 2015)				
θ	T_p	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T_p	T	*Gap Best*	*Gap Avg.*	# BKS
5	0.5	70	0.04	0.09	15	670	0.03	0.04	18	1009	0.02	0.04	18	66.1	158	0.03	0.08	16
6	1.2	58	0.05	0.09	18	553	0.05	0.06	19	833	0.05	0.06	19	94.3	168	0.06	0.11	17
7	1.6	52	0.00	0.05	19	490	0.00	0.00	20	737	0.00	0.00	20	113.6	183	0.00	0.04	20
8	3.0	48	0.00	0.02	19	446	0.00	0.01	19	665	0.00	0.01	19	204.7	259	0.00	0.02	20
9	4.4	45	0.00	0.00	20	409	0.00	0.00	20	609	0.00	0.00	20	264.1	315	0.00	0.01	20
10	6.0	43	0.01	0.05	19	375	0.00	0.02	20	563	0.00	0.02	20	511.0	561	0.00	0.02	20
11	7.6	42	0.01	0.03	19	347	0.01	0.03	19	514	0.01	0.03	19	357.7	403	0.00	0.01	20
12	9.8	43	0.01	0.02	19	333	0.01	0.01	19	494	0.01	0.01	19	974.2	1017	0.00	0.01	20
13	13.5	44	0.00	0.03	20	310	0.00	0.02	20	458	0.00	0.01	20	867.1	907	0.00	0.00	20
14	18.2	46	0.00	0.01	20	295	0.00	0.01	20	429	0.00	0.00	20	2283.6	2321	0.00	0.00	20
15	24.4	51	0.00	0.00	20	285	0.00	0.00	20	415	0.00	0.00	20	3090.4	3127	0.00	0.00	20
Total	8.2	49	0.01	0.04	208	410	0.01	0.02	214	611	0.01	0.02	214	802.4	856	0.01	0.03	213

Table 6: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (5, 0)$ and the `Golden` instances grouped by average cluster size; each group comprises 20 instances.

θ-values, while for more iterations the LMNS iterations primarily impact the overall time T.

For instances with small average cluster size ($\theta \leq 6$), $\text{LMNS}^{3,3}_{5\,000}$ produces slightly worse results but in shorter computation time compared to UHGS. For the example of $\theta = 5$, $\text{LMNS}^{3,3}_{5\,000}$ has a *Gap Best* of 0.04 % (47 seconds) compared to UHGS with a gap of 0.03 % (158 seconds). When accepting longer computation times, $\text{LMNS}^{3,3}_{50\,000}$ reduces *Gap Best* to 0.00 % (463 seconds). Both $\text{LMNS}^{3,3}_{50\,000}$ and $\text{LMNS}^{3,3}_{100\,000}$ outperform UHGS w.r.t. the gaps and the number of BKS produced. Similar results can be achieved for setting $(k_{\text{VND}}, k_{\text{LMNS}}) = (5, 0)$.

In general, if the cluster size θ is increased, LMNS tends to produce better results. Starting from $\theta = 5$ and 6 using $\text{LMNS}^{5,0}_{5\,000}$, we achieve a *Gap Best* not exceeding 0.05 %. For $\theta \geq 7$, *Gap Best* values not larger than 0.01 % result, and for $\theta \geq 13$ all BKS are found. The latter result holds also for $\text{LMNS}^{3,3}_{5\,000}$. Similarly, the average gap improves with the cluster size and both LMNS settings are able to find at least 19 BKS for $\theta \geq 7$ even with only 5 000 LMNS iterations. In comparison, also UHGS is able to find all BKS for $\theta \geq 7$, however its time consumption raises drastically for large clusters.

Overall, the comparison of LMNS and UHGS can be summarized as follows: First, LMNS produces slightly better CluVRP results in shorter computation times for instances with up to $n = 483$ customers. Second, although UHGS produces some smaller gaps on the large-scale instances ($n \geq 560$), LMNS is able to compute two new BKS for the benchmark set `Li` (and three in total). Third, for larger average cluster sizes ($\theta \geq 13$), the same high-quality results obtained with UHGS can be computed with LMNS with less effort.

5. Conclusions

In this paper, we proposed a new metaheuristic for the CluVRP. Our new LMNS approach can be classified as a LNS that uses multiple destroy and repair operators together with a VND-based local improvement procedure. An ILS-based preprocessing phase first computes all intra-cluster routes for every possible entry-exit combination. Then, for the actual LNS, we adapted four destroy and two repair operators to the case of the removal of clusters from and their subsequent insertion into CluVRP routes. Moreover, cluster neighborhoods that exchange clusters between routes in a classical manner similar to edge-exchange methods for CVRP have been implemented. A fundamental component that we developed is a new neighborhood specifically tailored to the CluVRP, i.e., a generalization of the Balas-Simonetti neighborhood that is able to simultaneously decide on the permutation of the clusters in each route as well as the entry-exit combinations. We have shown that although the generalized Balas-Simonetti neighborhood comprises exponentially many possible routes, it can be searched efficiently with an effort that grows only linearly with the number of clusters and linearly with the number of entry-exit combinations. Computational experiments have proven that the generalized Balas-Simonetti neighborhood is complementary to the cluster neighborhoods. This complementarity allows the detection of CluVRP solutions with a better quality in relatively shorter time than without the Balas-Simonetti neighborhood.

Although based on several components, the overall LMNS is clearly structured and only a few parameters had to be tuned in parameter studies. We have shown that none of the LNS destroy and repair operators is dispensable in the sense that when LMNS was run without one of the operators, the quality of solutions deteriorates. Weights that control the random selection of operators were chosen in a straightforward manner, since we found that the LMNS is not really sensitive w.r.t. these choices. The comparison with the exact algorithm of Battarra *et al.* (2014) reveals that, out of 230 instances, LMNS improved the solutions in seven cases (when the exact algorithm was prematurely terminated after 3 600 seconds) and computed 217 identical solutions. We also compared two versions of the LMNS against the UHGS metaheuristic of Vidal *et al.* (2015) that constitutes the state of the art for the CluVRP w.r.t. solution quality and computation times. The LMNS is competitive with the UHGS: Over all 242 benchmark instances, average computation times and gaps are in favor of LMNS compared to UHGS because setups with up to 50 000 LMNS iterations produce the same average gap of only 0.04 % and best gap of 0.01 %, but consume less computation time. Conversely, with more LMNS iterations, we arrived at similar computation times as UHGS but a smaller average gap of 0.03 % (identical best gap 0.01 %). Finally, three new best solutions were found with the LMNS.

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. (2015). Private communication.

Battarra, M. and Vidal, T. (2017). Private communication.

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.

Bode, C. and Irnich, S. (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, **60**(5), 1167–1182.

Chisman, J. A. (1975). The clustered traveling salesman problem. *Computers & Operations Research*, **2**(2), 115–119.

Christofides, N. (1970). The shortest hamiltonian chain of a graph. *SIAM Journal on Applied Mathematics*, **19**(4), 689–696.

Defryn, C. and Sörensen, K. (2015). A two-level variable neighbourhood search for the Euclidean clustered vehicle routing problem. Technical Report D/2015/1169/002, University of Antwerp, Faculty of Applied Economics, Antwerp, The Netherlands.

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.

Gutin, G., Yeo, A., and Zverovich, A. (2002). 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*, chapter 3, pages 207–222. Kluwer, Dordrecht.

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

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*, 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.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, **21**(2), 498–516.

Marc, A. H., Fuksz, L., Pop, P. C., and Dănciulescu, D. (2015). A novel hybrid algorithm for solving the clustered vehicle routing problem. In E. Onieva, I. Santos, E. Osaba, H. Quintián, and E. Corchado, editors, *Hybrid Artificial Intelligent Systems: 10th International Conference, HAIS 2015, Bilbao, Spain, June 22-24, 2015, Proceedings*, pages 679–689. Springer International Publishing, Cham.

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.

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 OptimizationFeature Cluster: New Opportunities for Operations Research.

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

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.

Vidal, T. (2016). Node, edge, arc routing and turn penalties: Multiple problems – one neighborhood extension. Technical report, Departamento de Informtica, Pontifcia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil. (Revised version of Technical Report from April 2015).

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, **60**(3), 611–624.

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

Appendix

This appendix is supposed to become online supplementary material.

I. Detailed Results

Tables 7–12 provide detailed results for each instance. Columns BKS and *First found by* show the best known solution and pointers to the literature (or our LMNS) where this solution was found first. Columns in the section LMNS show the best solution out of ten runs (Best), the average solution over ten runs (Avg.), the time consumption for the preprocessing T_p, and the average total time over ten runs T. All LMNS runs were performed with the setting $(k_{\text{VND}}, k_{\text{LMNS}}, It_{\text{LMNS}}) = (3, 3, 100\,000)$.

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
G	261	131	12	3693	Vidal *et al.* (2015)	3693	3712.1	0.1	281
C	100	51	5	642	Battarra *et al.* (2014)	642	642	0.1	83
C	120	61	4	807	Battarra *et al.* (2014)	807	807	0.1	215
C	150	76	6	816	Battarra *et al.* (2014)	816	816	0.1	211
C	199	100	8	955	Vidal *et al.* (2015)*	958	965	0.1	132
G	261	88	9	3281	LMNS**	3281	3283.4	0.1	473
C	100	34	4	607	Battarra *et al.* (2014)	607	607	0.1	100
C	120	41	3	691	Battarra *et al.* (2014)	691	691	0.1	158
C	150	51	4	804	Battarra *et al.* (2014)	804	804	0.1	181
C	199	67	6	908	Battarra *et al.* (2014)	908	908	0.1	347

Table 7: Detailed results for `GVRP` instances; * found by one of their ILS approaches, not by UHGS; ** found with LMNS and the setting $(k_{\text{VND}}, k_{\text{LMNS}}, It_{\text{LMNS}}) = (3, 3, 50\,000)$ (also with several other settings).

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
`Li`	560	113	39	27962	Vidal *et al.* (2015)	27962	27962	2.9	2663
`Li`	600	121	62	29051	Vidal *et al.* (2015)	29059	29078.2	2.6	3269
`Li`	640	129	10	21243	Vidal *et al.* (2015)	21243	21268.1	6.2	1827
`Li`	720	145	11	24486	Vidal *et al.* (2015)	24488	24516.1	6.8	2394
`Li`	760	153	78	35166	Vidal *et al.* (2015)	35173	35200.9	5.4	5230
`Li`	800	161	11	27238	Vidal *et al.* (2015)	27251	27293.9	5.0	3490
`Li`	840	169	86	37859	Vidal *et al.* (2015)	37863	37889.1	4.8	5943
`Li`	880	177	11	30483	Vidal *et al.* (2015)	30483	30551	8.9	4453
`Li`	960	193	11	32656	Vidal *et al.* (2015)	32668	32784.5	3.5	6118
`Li`	1040	209	11	35885	Vidal *et al.* (2015)	35915	35954.4	5.7	6345
`Li`	1120	225	11	38652	LMNS*	38706	38719.7	6.2	8190
`Li`	1200	241	11	41388	LMNS**	41431	41485.9	5.2	10946

Table 8: Detailed results for `Li` instances; * found with LMNS during experimentation; ** found with LMNS and the setting $(k_{\text{VND}}, k_{\text{LMNS}}, It_{\text{LMNS}}) = (1, 1, 5\,000)$.

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
Golden1	240	17	4	4831	Battarra *et al.* (2014)	4831	4831	15.7	173
Golden1	240	18	4	4847	Battarra *et al.* (2014)	4847	4847	4.3	183
Golden1	240	19	4	4872	Battarra *et al.* (2014)	4872	4872	4.3	187
Golden1	240	21	4	4889	Battarra *et al.* (2014)	4889	4889	3.9	198
Golden1	240	22	4	4908	Battarra *et al.* (2014)	4908	4908	3.6	209
Golden1	240	25	4	4899	Battarra *et al.* (2014)	4899	4899	3.9	225
Golden1	240	27	4	4934	Battarra *et al.* (2014)	4934	4934	2.2	244
Golden1	240	31	4	5050	Battarra *et al.* (2014)	5050	5050	1.2	277
Golden1	240	35	4	5102	Battarra *et al.* (2014)	5102	5102	0.6	326
Golden1	240	41	4	5097	Battarra *et al.* (2014)	5097	5097	0.4	354
Golden1	240	49	4	5000	Battarra *et al.* (2014)	5000	5000	0.3	447
Golden2	320	22	4	7716	Battarra *et al.* (2014)	7716	7716	19.4	325
Golden2	320	23	4	7693	Battarra *et al.* (2014)	7693	7693	19.4	334
Golden2	320	25	4	7668	Battarra *et al.* (2014)	7668	7668	19.1	347
Golden2	320	27	4	7638	Battarra *et al.* (2014)	7638	7638	10.4	360
Golden2	320	30	4	7617	Battarra *et al.* (2014)	7617	7617	4.3	373
Golden2	320	33	4	7640	Battarra *et al.* (2014)	7640	7640	2.7	408
Golden2	320	36	4	7643	Battarra *et al.* (2014)	7643	7643	2.4	431
Golden2	320	41	4	7738	Battarra *et al.* (2014)	7738	7738	2.1	482
Golden2	320	46	4	7861	Battarra *et al.* (2014)	7861	7861	1.1	545
Golden2	320	54	4	7920	Battarra *et al.* (2014)	7920	7920	1.2	641
Golden2	320	65	4	7892	Battarra *et al.* (2014)	7892	7893.6	1.0	812
Golden3	400	27	4	10540	Battarra *et al.* (2014)	10540	10540	91.5	581
Golden3	400	29	4	10504	Battarra *et al.* (2014)	10504	10504	36.6	598
Golden3	400	31	4	10486	Battarra *et al.* (2014)	10486	10486	12.8	606
Golden3	400	34	4	10465	Battarra *et al.* (2014)	10465	10465	12.3	614
Golden3	400	37	4	10482	Battarra *et al.* (2014)	10482	10482	12.5	662
Golden3	400	41	4	10501	Battarra *et al.* (2014)	10501	10501	11.1	710
Golden3	400	45	4	10485	Battarra *et al.* (2014)	10485	10485	7.9	786
Golden3	400	51	4	10583	Battarra *et al.* (2014)	10583	10583	3.2	853
Golden3	400	58	4	10776	Battarra *et al.* (2014)	10776	10776	1.8	925
Golden3	400	67	4	10797	Battarra *et al.* (2014)	10797	10797	1.7	1142
Golden3	400	81	4	10614	Battarra *et al.* (2014)	10614	10614	1.7	1440
Golden4	480	33	4	13598	Battarra *et al.* (2014)	13598	13598	56.7	746
Golden4	480	35	4	13643	Battarra *et al.* (2014)	13643	13643	55.4	765
Golden4	480	37	4	13520	Battarra *et al.* (2014)	13520	13520	25.8	767
Golden4	480	41	4	13460	Battarra *et al.* (2014)	13460	13460	24.4	843
Golden4	480	44	4	13568	Battarra *et al.* (2014)	13568	13568	24.1	880
Golden4	480	49	4	13758	Battarra *et al.* (2014)	13758	13758	23.9	953
Golden4	480	54	4	13760	Battarra *et al.* (2014)	13760	13760	22.8	1007
Golden4	480	61	4	13791	Battarra *et al.* (2014)	13791	13791	22.8	1141
Golden4	480	69	4	13966	Battarra *et al.* (2014)	13966	13966.6	5.7	1261
Golden4	480	81	4	13975	Battarra *et al.* (2014)	13975	13975	2.6	1470
Golden4	480	97	4	13775	Battarra *et al.* (2014)	13775	13779	1.7	1883
Golden5	200	14	4	7622	Battarra *et al.* (2014)	7622	7622	18.1	100
Golden5	200	15	3	7424	Battarra *et al.* (2014)	7424	7424	16.8	95
Golden5	200	16	3	7491	Battarra *et al.* (2014)	7491	7491	16.5	105
Golden5	200	17	3	7434	Battarra *et al.* (2014)	7434	7434	15.0	94
Golden5	200	19	4	7576	Battarra *et al.* (2014)	7576	7576	6.2	105
Golden5	200	21	4	7596	Battarra *et al.* (2014)	7596	7596	4.4	117
Golden5	200	23	4	7643	Battarra *et al.* (2014)	7643	7643	4.6	147
Golden5	200	26	4	7560	Battarra *et al.* (2014)	7560	7560	4.4	164
Golden5	200	29	4	7410	Battarra *et al.* (2014)	7410	7410	4.3	157
Golden5	200	34	4	7429	Battarra *et al.* (2014)	7429	7429	3.1	195
Golden5	200	41	4	7241	Battarra *et al.* (2014)	7241	7241	0.4	280

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

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
Golden6	280	19	3	8624	Battarra *et al.* (2014)	8624	8624	58.0	228
Golden6	280	21	3	8628	Battarra *et al.* (2014)	8628	8628	48.6	229
Golden6	280	22	3	8646	Battarra *et al.* (2014)	8646	8646	29.2	223
Golden6	280	24	4	8853	Battarra *et al.* (2014)	8853	8853	18.0	251
Golden6	280	26	4	8910	Battarra *et al.* (2014)	8910	8910	18.0	270
Golden6	280	29	4	8936	Battarra *et al.* (2014)	8936	8936	5.7	358
Golden6	280	32	4	8891	Battarra *et al.* (2014)	8891	8891	3.8	381
Golden6	280	36	4	8969	Battarra *et al.* (2014)	8969	8969	3.7	374
Golden6	280	41	4	9028	Battarra *et al.* (2014)	9028	9028	3.8	423
Golden6	280	47	4	8923	Battarra *et al.* (2014)	8923	8923	3.6	495
Golden6	280	57	4	9028	Battarra *et al.* (2014)	9028	9028	1.0	644
Golden7	360	25	3	9904	Battarra *et al.* (2014)	9904	9904	56.8	409
Golden7	360	26	3	9888	Battarra *et al.* (2014)	9888	9888	39.8	413
Golden7	360	28	3	9917	Battarra *et al.* (2014)	9917	9917	38.4	409
Golden7	360	31	4	10021	Battarra *et al.* (2014)	10021	10021	28.5	492
Golden7	360	33	4	10029	Battarra *et al.* (2014)	10029	10029	21.3	508
Golden7	360	37	4	10131	Battarra *et al.* (2014)	10131	10131	20.7	551
Golden7	360	41	4	10052	Battarra *et al.* (2014)	10052	10052	20.7	661
Golden7	360	46	4	10080	Battarra *et al.* (2014)	10080	10080	6.8	701
Golden7	360	52	4	10095	Battarra *et al.* (2014)	10095	10095	1.2	768
Golden7	360	61	4	10096	Battarra *et al.* (2014)	10096	10096	1.1	858
Golden7	360	73	4	10014	Battarra *et al.* (2014)	10014	10014	1.2	1139
Golden8	440	30	4	10866	Battarra *et al.* (2014)	10866	10866	31.0	599
Golden8	440	32	4	10831	Battarra *et al.* (2014)	10831	10831	31.2	638
Golden8	440	34	4	10847	Battarra *et al.* (2014)	10847	10847	31.1	664
Golden8	440	37	4	10859	Battarra *et al.* (2014)	10859	10859	27.4	678
Golden8	440	41	4	10934	Battarra *et al.* (2014)	10934	10934	27.0	700
Golden8	440	45	4	10960	Battarra *et al.* (2014)	10960	10960	26.4	769
Golden8	440	49	4	11042	Battarra *et al.* (2014)	11042	11042	5.9	826
Golden8	440	56	4	11194	Battarra *et al.* (2014)	11194	11194.3	2.9	997
Golden8	440	63	4	11252	Battarra *et al.* (2014)	11252	11252	2.8	993
Golden8	440	74	4	11321	Battarra *et al.* (2014)	11321	11321	2.8	1249
Golden8	440	89	4	11209	Battarra *et al.* (2014)	11209	11209.4	1.5	1652
Golden9	255	18	4	300	Battarra *et al.* (2014)	300	300	4.4	185
Golden9	255	19	4	299	Battarra *et al.* (2014)	299	299	3.9	171
Golden9	255	20	4	296	Battarra *et al.* (2014)	296	296	3.2	203
Golden9	255	22	4	290	Battarra *et al.* (2014)	290	290	2.4	210
Golden9	255	24	4	290	Battarra *et al.* (2014)	290	290	1.9	222
Golden9	255	26	4	288	Battarra *et al.* (2014)	288	288	1.3	232
Golden9	255	29	4	292	Battarra *et al.* (2014)	292	292	0.8	256
Golden9	255	32	4	297	Battarra *et al.* (2014)	297	297	0.7	272
Golden9	255	37	4	294	Battarra *et al.* (2014)	294	294	0.6	317
Golden9	255	43	4	295	Battarra *et al.* (2014)	295	295.6	0.4	367
Golden9	255	52	4	296	Battarra *et al.* (2014)	296	296.9	0.1	432
Golden10	323	22	4	367	Battarra *et al.* (2014)	367	367	6.6	234
Golden10	323	24	4	361	Battarra *et al.* (2014)	361	361	3.7	232
Golden10	323	25	4	359	Battarra *et al.* (2014)	359	360.2	3.2	249
Golden10	323	27	4	361	Battarra *et al.* (2014)	361	361	3.0	274
Golden10	323	30	4	367	Battarra *et al.* (2014)	368	368	2.5	290
Golden10	323	33	4	373	Battarra *et al.* (2014)	375	375	1.8	302
Golden10	323	36	4	385	Battarra *et al.* (2014)	385	385.3	1.2	314
Golden10	323	41	4	400	Battarra *et al.* (2014)	400	400	0.5	330
Golden10	323	47	4	398	Battarra *et al.* (2014)	398	398	0.5	366
Golden10	323	54	4	393	Battarra *et al.* (2014)	393	393.4	0.4	418
Golden10	323	65	4	387	Battarra *et al.* (2014)	387	387.6	0.3	504

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

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
Golden11	399	27	5	457	Battarra *et al.* (2014)	457	457	5.4	452
Golden11	399	29	5	455	Battarra *et al.* (2014)	455	455	5.0	472
Golden11	399	31	5	455	Battarra *et al.* (2014)	455	455	4.2	467
Golden11	399	34	5	455	Battarra *et al.* (2014)	455	455	3.0	547
Golden11	399	37	5	459	Battarra *et al.* (2014)	459	459	2.4	553
Golden11	399	40	5	461	Battarra *et al.* (2014)	461	461	1.3	586
Golden11	399	45	5	462	Battarra *et al.* (2014)	462	462	1.1	658
Golden11	399	50	5	458	Battarra *et al.* (2014)	458	458	1.0	674
Golden11	399	58	5	456	Battarra *et al.* (2014)	456	456	0.8	734
Golden11	399	67	5	454	Battarra *et al.* (2014)	454	454	0.5	896
Golden11	399	80	5	451	Battarra *et al.* (2014)	451	451	0.2	1028
Golden12	483	33	5	535	Battarra *et al.* (2014)	535	535	5.4	595
Golden12	483	35	5	537	Battarra *et al.* (2014)	537	538.2	5.0	619
Golden12	483	38	5	535	Battarra *et al.* (2014)	535	538.6	3.7	652
Golden12	483	41	5	537	Battarra *et al.* (2014)	537	537	2.5	682
Golden12	483	44	5	535	Battarra *et al.* (2014)	535	536.4	2.4	715
Golden12	483	49	5	533	Battarra *et al.* (2014)	533	533.4	2.3	810
Golden12	483	54	5	535	Battarra *et al.* (2014)	535	535	1.9	859
Golden12	483	61	5	535	Vidal *et al.* (2015)	535	535	1.4	932
Golden12	483	70	5	533	Vidal *et al.* (2015)	533	533	1.2	1042
Golden12	483	81	5	535	Vidal *et al.* (2015)	535	535.2	0.5	1163
Golden12	483	97	5	544	Vidal *et al.* (2015)	544	544	0.1	1432
Golden13	252	17	4	552	Battarra *et al.* (2014)	552	552	3.2	158
Golden13	252	19	4	549	Battarra *et al.* (2014)	549	549	2.1	186
Golden13	252	20	4	548	Battarra *et al.* (2014)	548	548	2.0	209
Golden13	252	22	4	548	Battarra *et al.* (2014)	548	548	1.7	224
Golden13	252	23	4	548	Battarra *et al.* (2014)	548	548	1.6	231
Golden13	252	26	4	542	Battarra *et al.* (2014)	542	542	1.1	250
Golden13	252	29	4	540	Battarra *et al.* (2014)	540	540	0.7	292
Golden13	252	32	4	543	Battarra *et al.* (2014)	543	543	0.6	289
Golden13	252	37	4	545	Battarra *et al.* (2014)	545	545	0.3	297
Golden13	252	43	4	553	Battarra *et al.* (2014)	553	553	0.1	381
Golden13	252	51	4	560	Battarra *et al.* (2014)	560	560	0.1	449
Golden14	320	22	4	692	Battarra *et al.* (2014)	692	692	4.9	267
Golden14	320	23	4	688	Battarra *et al.* (2014)	688	688	3.7	266
Golden14	320	25	4	678	Battarra *et al.* (2014)	678	678	2.9	255
Golden14	320	27	4	676	Battarra *et al.* (2014)	676	676	2.3	275
Golden14	320	30	4	678	Battarra *et al.* (2014)	678	678	1.7	313
Golden14	320	33	4	682	Battarra *et al.* (2014)	682	682	1.6	349
Golden14	320	36	4	687	Battarra *et al.* (2014)	687	687	0.8	345
Golden14	320	41	4	690	Battarra *et al.* (2014)	690	690	0.5	414
Golden14	320	46	4	694	Battarra *et al.* (2014)	694	694	0.3	453
Golden14	320	54	4	699	Battarra *et al.* (2014)	699	699	0.1	509
Golden14	320	65	4	703	Battarra *et al.* (2014)	703	703	0.1	655
Golden15	396	27	4	842	Battarra *et al.* (2014)	842	842	5.3	359
Golden15	396	29	4	843	Battarra *et al.* (2014)	843	843	4.5	385
Golden15	396	31	4	837	Battarra *et al.* (2014)	837	837	3.3	379
Golden15	396	34	4	838	Battarra *et al.* (2014)	838	838	2.3	404
Golden15	396	37	4	845	Battarra *et al.* (2014)	845	845	1.8	397
Golden15	396	40	4	849	Battarra *et al.* (2014)	849	849	1.3	465
Golden15	396	45	5	853	Battarra *et al.* (2014)	853	853	0.9	518
Golden15	396	50	5	851	Battarra *et al.* (2014)	851	851	0.7	562
Golden15	396	57	5	850	Battarra *et al.* (2014)	850	850	0.5	638
Golden15	396	67	5	855	Battarra *et al.* (2014)	855	855.6	0.1	688
Golden15	396	80	5	857	Battarra *et al.* (2014)	857	857.6	0.1	920

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

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
Golden16	480	33	5	1030	Battarra *et al.* (2014)	1030	1030	6.7	656
Golden16	480	35	5	1028	Battarra *et al.* (2014)	1028	1028	5.0	661
Golden16	480	37	5	1028	Battarra *et al.* (2014)	1028	1028	3.7	704
Golden16	480	41	5	1032	Battarra *et al.* (2014)	1032	1032	2.6	759
Golden16	480	44	5	1028	Battarra *et al.* (2014)	1028	1028	2.0	756
Golden16	480	49	5	1031	Battarra *et al.* (2014)	1031	1031	1.4	822
Golden16	480	54	5	1022	Battarra *et al.* (2014)	1022	1022	1.3	914
Golden16	480	61	5	1013	Battarra *et al.* (2014)	1014	1014	1.0	1046
Golden16	480	69	5	1012	Battarra *et al.* (2014)	1012	1012	0.8	1203
Golden16	480	81	5	1018	Battarra *et al.* (2014)	1018	1018	0.3	1376
Golden16	480	97	5	1018	Battarra *et al.* (2014)	1019	1019.6	0.1	1594
Golden17	240	17	3	418	Battarra *et al.* (2014)	418	418	6.9	196
Golden17	240	18	3	419	Battarra *et al.* (2014)	419	419	5.9	215
Golden17	240	19	3	422	Battarra *et al.* (2014)	422	422	4.8	213
Golden17	240	21	3	425	Battarra *et al.* (2014)	425	425	4.1	220
Golden17	240	22	3	424	Battarra *et al.* (2014)	424	424	3.9	215
Golden17	240	25	3	418	Battarra *et al.* (2014)	418	418	1.8	250
Golden17	240	27	3	414	Battarra *et al.* (2014)	414	414	1.3	250
Golden17	240	31	4	421	Battarra *et al.* (2014)	421	421	0.5	308
Golden17	240	35	4	417	Battarra *et al.* (2014)	417	417	0.2	322
Golden17	240	41	4	412	Battarra *et al.* (2014)	412	412	0.1	400
Golden17	240	49	4	414	Battarra *et al.* (2014)	414	414	0.1	473
Golden18	300	21	4	592	Battarra *et al.* (2014)	592	592	14.7	231
Golden18	300	22	4	594	Battarra *et al.* (2014)	594	594	14.1	245
Golden18	300	24	4	592	Battarra *et al.* (2014)	592	592	14.4	253
Golden18	300	26	4	590	Battarra *et al.* (2014)	590	590	6.4	303
Golden18	300	28	4	577	Battarra *et al.* (2014)	577	577	3.1	364
Golden18	300	31	4	578	Battarra *et al.* (2014)	578	578	2.2	365
Golden18	300	34	4	582	Battarra *et al.* (2014)	582	582	1.5	383
Golden18	300	38	4	586	Battarra *et al.* (2014)	586	586	1.1	407
Golden18	300	43	4	594	Battarra *et al.* (2014)	594	594	0.5	427
Golden18	300	51	4	601	Battarra *et al.* (2014)	601	601	0.1	521
Golden18	300	61	4	599	Battarra *et al.* (2014)	599	599	0.1	659
Golden19	360	25	10	925	Battarra *et al.* (2014)	925	925	35.6	352
Golden19	360	26	10	924	Battarra *et al.* (2014)	924	924	29.4	365
Golden19	360	28	4	808	Battarra *et al.* (2014)	808	808	21.2	327
Golden19	360	31	4	811	Battarra *et al.* (2014)	812	812	9.9	391
Golden19	360	33	4	797	Battarra *et al.* (2014)	797	797	4.5	426
Golden19	360	37	5	799	Battarra *et al.* (2014)	799	799	1.9	490
Golden19	360	41	5	789	Battarra *et al.* (2014)	789	789	1.1	592
Golden19	360	46	5	788	Battarra *et al.* (2014)	788	788	1.0	625
Golden19	360	52	5	800	Battarra *et al.* (2014)	800	800	0.9	691
Golden19	360	61	5	807	Battarra *et al.* (2014)	807	807	0.6	769
Golden19	360	73	5	810	Battarra *et al.* (2014)	810	810	0.4	953
Golden20	420	29	11	1220	Battarra *et al.* (2014)	1220	1220	43.1	477
Golden20	420	31	12	1232	Battarra *et al.* (2014)	1232	1232	28.3	488
Golden20	420	33	12	1208	Battarra *et al.* (2014)	1208	1208	25.9	514
Golden20	420	36	5	1059	Battarra *et al.* (2014)	1059	1059	15.0	485
Golden20	420	39	5	1052	Battarra *et al.* (2014)	1052	1052	7.8	523
Golden20	420	43	5	1052	Battarra *et al.* (2014)	1052	1052	4.5	543
Golden20	420	47	5	1053	Battarra *et al.* (2014)	1053	1053	4.5	646
Golden20	420	53	5	1058	Battarra *et al.* (2014)	1058	1058	4.4	689
Golden20	420	61	5	1058	Battarra *et al.* (2014)	1058	1058	4.3	828
Golden20	420	71	5	1049	Battarra *et al.* (2014)	1059	1059	3.7	984
Golden20	420	85	5	1049	Battarra *et al.* (2014)	1049	1049	0.7	1119

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