On Clustered Vehicle-Routing Problems

Dissertation
zur Erlangung des Grades eines Doktors der
wirtschaftlichen Staatswissenschaften
(Dr. rer. pol.)
des Fachbereichs Rechts- und Wirtschaftswissenschaften
der Johannes Gutenberg-Universität Mainz

vorgelegt von
M. Sc. Timo Hintsch
in Mainz

im Jahre 2019

Tag der mündlichen Prüfung: 19.06.2019

Danksagung

Zuerst danke ich meinem Doktorvater ganz herzlich für die sowohl fachlich als auch menschlich großartige Zusammenarbeit. Von seiner Expertise habe ich bei der Erstellung dieser Arbeit sehr profitiert. Seine Offenheit und Art der Lehrstuhlleitung sorgen (auch außerhalb des Arbeitsalltags) für eine tolle und freundschaftliche Atmosphäre. An dieser hat auch die Sekretärin des Lehrstuhls großen Anteil. Ihr sowie all meinen Lehrstuhlkolleginnen und -kollegen gilt mein Dank dafür, dass sie die Zeit am Lehrstuhl so schön gemacht haben. Insbesondere danke ich meinem 'Mastervater' für zahlreiche Hilfestellungen und (nicht nur fachliche) Diskussionen, aus denen ich viel gelernt habe; meinem Bürokollegen für seine detaillierten und sehr hilfreichen Ratschläge; sowie meinem kanadischen Freund, dessen Fachwissen mir ebenfalls oft weiterhalf.

Des Weiteren bedanke ich mich bei meinen Eltern, die mich zum Schritt der Promotion ermutigt haben und mir bereits auf meinem ganzen Lebensweg in jeglichen Situationen durch Rat und vor allem Tat zur Seite stehen. Zudem danke ich meinen Freunden inkl. meines Bruders für viele abwechslungsreiche außeruniversitäre Aktivitäten, durch die ich auch in schwierigen Phasen wieder aufgemuntert wurde.

Darüber hinaus danke ich meiner lieben Freundin sehr herzlich für ihre Geduld und Unterstützung, insbesondere in der Endphase meiner Dissertation und im Rahmen der Disputation.

Contents

List of Papers

- Timo Hintsch[1], Prof. Dr. Stefan Irnich[1] (2018). Large multiple neighborhood search for the clustered vehicle-routing problem. *European Journal of Operational Research* 270 (1), pp. 118–131.
- Timo Hintsch, Prof. Dr. Stefan Irnich (2020). Exact solution of the soft-clustered vehicle-routing problem. *European Journal of Operational Research* 280 (1), pp. 164–178.
- Timo Hintsch (2019). Large Multiple Neighborhood Search for the Soft-Clustered Vehicle-Routing Problem. Technical Report LM-2019-01, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany, available online at `http://logistik.bwl.uni-mainz.de/158.php`. *submitted to Computers & Operations Research.*

[1]Johannes Gutenberg-Universität Mainz, Lehrstuhl für BWL insb. Logistikmanagement, Jakob-Welder-Weg 9, D-55128 Mainz, Germany

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Routing problems arise in numerous real-life scenarios, for example transportation, telecommunication, ship routing, postal and freight delivery, waste collection, and winter services. The *vehicle-routing problem* (VRP, Toth and Vigo, 2014) is one of the most important routing problems. It attracts interest of researchers since Dantzig and Ramser (1959) introduced the problem 60 years ago. In general, the VRP searches for a set of routes that is performed by a given fleet of vehicles to fulfill a set of requests with minimal total routing costs. The most studied version is the *capacitated vehicle-routing problem* (CVRP), in which homogeneous vehicles deliver goods from a single depot to customers and the capacity of the vehicles is limited (Irnich *et al.*, 2014). Manifold variants of the CVRP exist in the literature and were surveyed by Irnich *et al.* (2014). For instance, the *pickup-and-delivery problem* (Parragh *et al.*, 2008; Battarra *et al.*, 2014b) considers the transportation of goods or passengers from different pickup points to different delivery points.

The thesis at hand tackles two versions of the *clustered vehicle-routing problem* (CluVRP, Sevaux and Sörensen, 2008) in which the customers are partitioned into clusters. Both versions are introduced in the next Section 1.1. Afterwards, Section 1.2 briefly explains the solution methods that are used in this thesis. An overview of both exact and heuristic solution methods for VRP problems is provided by Toth and Vigo (2014, Chapters 2–4). The contributions and the outline of this thesis are described in Section 1.3.

1.1 Clustered Vehicle-Routing Problems

The *clustered vehicle-routing problem* (CluVRP, Sevaux and Sörensen, 2008) assumes that all customers are grouped into disjoint clusters. In the basic version, a feasible CluVRP route must serve each cluster integrally, that is, all customers of the same cluster must be served by the same vehicle and in consecutive visits (*hard-cluster constraints*). Hence, visits to customers of the same cluster must not be interrupted by visits to customers of another cluster. The *soft-clustered vehicle-routing problem* (SoftCluVRP, Defryn and Sörensen, 2017) is a relaxation

of the CluVRP. Customers of the same cluster must still be visited by the same vehicle, but in contrast to the CluVRP, vehicles are allowed to enter and leave clusters several times (*soft-cluster constraint*). Hence, visits to customers of the same cluster may or may not be interrupted by visits to customers of another cluster. The two variants should not be mixed up with the *generalized vehicle-routing problem*, in which only one customer per cluster needs to be visited (Ghiani and Improta, 2000).

Both the SoftCluVRP as well as the CluVRP arise in practical scenarios where the routing decision must respect already taken clustering decisions. An example is parcel/small-package delivery in courier companies, where a large number of customers is divided into regional zones (Sevaux and Sörensen, 2008). In a first step, parcels are sorted into containers according to a given districting, e.g. by ZIP codes (see Butsch *et al.*, 2014, for districting). The districting, i.e. the sorting policy, is tackled on a tactical planning level and typically altered only once in a while. Note that the sorting policy must always be fixed before the actual demand distribution over the zones is known. In a second step, on the operational level, the filled containers are loaded to vehicles and the parcels are delivered to the recipients. In the CluVRP each parcel from one container has to be delivered before delivering parcels from another container is allowed, while in the SoftCluVRP there are no such requirements.

1.2 Solution Methods

Branch-and-price algorithms (Barnhart *et al.*, 1998) can be considered as a leading exact solution method for a wide range of VRP variants. It is a combination of branch-and-bound (Dakin, 1965) and column-generation (Lübbecke and Desrosiers, 2005; Desaulniers *et al.*, 2005). In column-generation, a route/path-based extended formulation of the considered VRP variant is restricted to a small subset of routes, resulting in the so-called *restricted master program* (RMP). Missing routes are dynamically and iteratively generated with the help of a pricing subproblem. In addition, branch-and-price algorithms relax the integrality constraints on the route variables in the RMP and enforce integrality later via branching. Similar approaches, a branch-and-cut and a branch-and-price-and-cut, were proposed for the CluVRP by (Battarra *et al.*, 2014a). For the SoftCluVRP, we are not aware of any exact solution method and we will present a branch-and-price algorithm for this problem variant in this thesis.

However, exact solution methods are usually limited to rather small instances and cannot generate solutions for practical scenarios in reasonable computation time. Hence, (meta-)heuristics, which do not guarantee optimality, are developed to provide good solutions in adequate time. Well-known examples are tabu search,

iterated local search, variable neighborhood search (VNS), simulated annealing, population-based algorithms, and several hybrids. Furthermore, large neighborhood search (Shaw, 1998; Ropke and Pisinger, 2006b) has been shown to be a powerful metaheuristic for many routing problems (Pisinger and Ropke, 2010). After constructing a feasible starting solution, the basic approach iteratively destroys parts of the current solution with a destroy operator and repairs the partial solution with a repair operator. As an extension, *large multiple neighborhood search* (LMNS, Pisinger and Ropke, 2007) uses multiple destroy and repair operators. In each iteration, both the destroy and the repair operator are chosen randomly.

For the CluVRP, the most impressive results for benchmark instances were obtained by a hybrid genetic algorithm called unified hybrid genetic search (UHGS, Vidal *et al.*, 2015) that uses the pre-computation of shortest Hamiltonian paths inside each cluster for each pair of customers belonging to that cluster. To the best of our knowledge, Defryn and Sörensen (2017) is the only article that addresses the SoftCluVRP. They presented a two-level VNS that was originally developed for the CluVRP in the same article. In the thesis at hand, we will present an LMNS for both of the problem variants.

1.3 Contributions and Outline

This thesis by publication consists of three papers that all contribute to the rather scarce (Soft)CluVRP literature and have either been published in or submitted to scientific journals. The main goal of the thesis is to develop effective metaheuristics for both the CluVRP as well as the SoftCluVRP and to provide the first exact solution method for the SoftCluVRP. In the following, we describe the structure of the thesis and the contributions of the individual chapters.

In Chapter 2, we present a new LMNS approach for the heuristic solution of the CluVRP. A novelty is the combination of multiple destroy and repair operators together with several neighborhoods that are used for post-optimizing the restored solutions. Furthermore, we exploit the hard-cluster constraints in different ways. All destroy and repair operators as well as all neighborhoods work on the cluster level and are based on the pre-computation of intra-cluster routes and a meta-representation of routes on a cluster level. The most remarkable exploitation is the generalization of the Balas-Simonetti neighborhood (Balas, 1999; Balas and Simonetti, 2001), which modifies the intra-cluster routings and the sequence of clusters in a route simultaneously. All neighborhoods, including the generalized Balas-Simonetti neighborhood, are combined in a variable neighborhood descent (VND, Hansen and Mladenović, 2001) for post-optimization.

In Chapter 3, we design and analyze different branch-and-price algorithms for the exact solution of the SoftCluVRP. The algorithms differ in the way the column-

generation subproblem is solved. As in many other VRP variants, the pricing problem can be formulated as a shortest-path problem with resource constraints (SPPRC, Irnich and Desaulniers, 2005), which is typically solved via dynamic-programming labeling algorithms. We provide such labeling algorithms that use all available state-of-the-art acceleration techniques. As an alternative, we model the subproblem and solve it with a branch-and-cut algorithm. Computational results show that this relatively simple integer programming-based approach clearly outperforms sophisticated dynamic-programming labeling algorithms.

For the heuristic solution of the SoftCluVRP, Chapter 4 adapts the LMNS from Chapter 2. Due to soft-cluster constraints the aforementioned exploitations are not applicable. Instead, new destroy and repair operators as well as cluster neighborhoods are tailored to the SoftCluVRP. Furthermore, a second VND is used for post-optimization, working on single routes and including the standard version of the Balas-Simonetti neighborhood.

Computational experiments (in both Chapters 2 and 4) show that both our LMNS metaheuristics compare favorably to existing approaches from the literature. Moreover, we generate many new best known solutions for benchmark instances, in particular in the case of the SoftCluVRP.

Finally, the thesis is summarized and concluded in Chapter 5.

Chapter 2

Large Multiple Neighborhood Search for the Clustered Vehicle-Routing Problem

Timo Hintsch, Stefan Irnich

Abstract

The clustered vehicle-routing problem is a variant of the classical capacitated vehicle-routing problem 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 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.

2.1 Introduction

The *clustered vehicle-routing problem* (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.

The 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 chapter, 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 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.

The CluVRP should not be mixed up with the *CluVRP with soft cluster constraints* introduced by Defryn and Sörensen (2017). In the latter problem, vehicles are allowed to enter and leave clusters several times while still all customers of a visited cluster must be served by the same vehicle. Hence, the CluVRP with soft cluster constraints is a relaxation of the problem considered in this chapter.

Note that the fundamental components of the LMNS, i.e., pre-computed ATSP solutions and the generalized Balas-Simonetti neighborhood preserve hard cluster constraints, so that the LMNS is not directly applicable to the CluVRP with soft cluster constraints. The latter problem certainly requires a completely different solution approach.

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 $V \setminus \{0\} = \{1, \ldots, n\}$ 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* $V_1, V_2, \ldots, V_{N-1}, V_N$. For the sake of convenience, we also define the *depot cluster* as $V_0 = \{0\}$. All customer clusters V_h, $h \in \{1, 2, \ldots, N\}$, 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 chapter is structured as follows. Section 2.2 reviews heuristic and exact CluVRP approaches from the literature. In Section 2.3, 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 2.4. Here, we also compare our computational results with the state-of-the-art metaheuristics for the CluVRP. Final conclusions are drawn in Section 2.5.

2.2 Literature Review

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.* (2014a) 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 (2017) 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 (2017) 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.*, 2014a) 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.

2.3 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 2.3.1, followed by the description of the new Balas-Simonetti neighborhood for clusters described in Section 2.3.2. Neighborhoods that allow the exchange of clusters between different routes are presented in Section 2.3.3. The destroy and repair operator of the LMNS and the overall algorithm are described in Sections 2.3.4 and 2.3.5.

2.3.1 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 $x_h(e_h, f_h) = (e_h, \ldots, f_h)$ 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 and a combined ILS/VND (both described in the following).

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 (see also Gutin *et al.*, 2007). 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 2.3.2.

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 $x = (x_0, x_1, x_2, \ldots, x_n, x_{n+1})$ be an ATSP tour or a Hamiltonian path. Each $x' = (x_{\pi(0)}, x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}, x_{\pi(n+1)})$ is a neighbor of x in $\mathcal{N}_k^{BS}$, if the permutation π of $\{0, 1, \ldots, n, n+1\}$ fulfills $\pi(0) = 0$, $\pi(n+1) = n+1$, and the following condition: if $i + k \leq j$ for a pair of indices $i, j \in \{1, 2, \ldots, n\}$, 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 2.1 for $k = 3$ and an original tour $x = (x_0, x_1, \ldots, x_n, x_{n+1})$ 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 $0 \leq i \leq n+1$. 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. As first done in (Irnich, 2008), 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.

The arc set of G_k^* is also well-defined: 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 $k(k+1)2^{k-2}$. 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 2.1, the states that can be reached with some s_0-s_{n+1}-path are depicted as solid (blue and red) vertices while unreachable states are depicted as dotted (gray) vertices. 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 vertices 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 $s \in W_i$. For the construction of G_k^*, we use the computer code (written in `C`) available on Neil Simonetti's website `http://www.andrew.cmu.edu/user/neils/`.

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 $i \in \{0, 1, \ldots, n\}$ 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 2.1). 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 2.1 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.

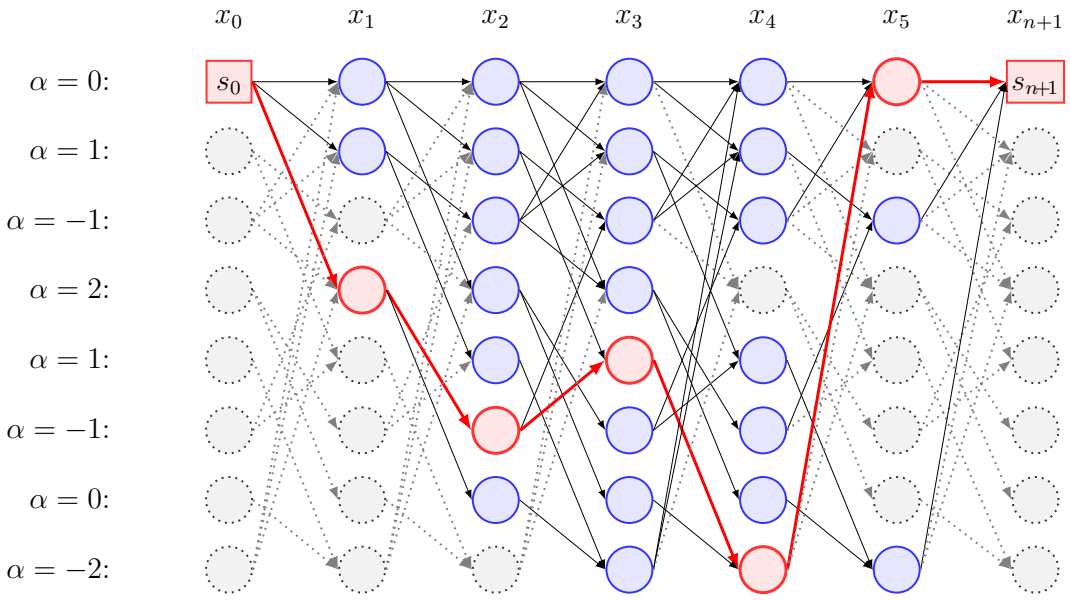

Figure 2.1: Auxiliary graph G_k^* for $k = 3$, current solution $x = (x_0, x_1, x_2, x_3, x_4, x_5, x_{n+1})$, and neighbor $x' = (x_0, x_3, x_1, x_4, x_2, x_5, x_{n+1}) \in \mathcal{N}_3^{BS}(x)$ implied by the highlighted s_0-s_{n+1}-path.

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 (more than $(k/e)^{n-1}$ neighbors for $n > k(k+1)$, see Gutin *et al.*, 2007, 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.

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, three 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). The four neighborhoods are applied in the order 2-opt, Or-opt, double bridge, and Balas-Simonetti. Note that all three classical edge-exchange neighborhoods can be searched efficiently in $\mathcal{O}({\lambda_h}^2)$ 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.

2.3.2 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 show that still a best combination, i.e., a best neighbor route, can be determined efficiently. For a general introduction to polynomially searchable exponentially-size neighborhoods we refer to (Gutin *et al.*, 2007).

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 2.3.1). The cost of such a route $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_p, \sigma_{p+1})$ is given by

$$c(\sigma) = \sum_{i=0}^{p} c_{f_i, e_{i+1}} + \sum_{i=1}^{p} \hat{c}_{e_i, f_i}. \tag{2.1}$$

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:

(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 $\{0, 1, \ldots, p, p+1\}$ with $\pi(0) = \pi(p+1) = 0$, and if $g + k \leq h$ for a pair of indices $g, h \in \{1, 2, \ldots, p\}$ 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 a best 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 2.2. 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 2.3.1). 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 2.2 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

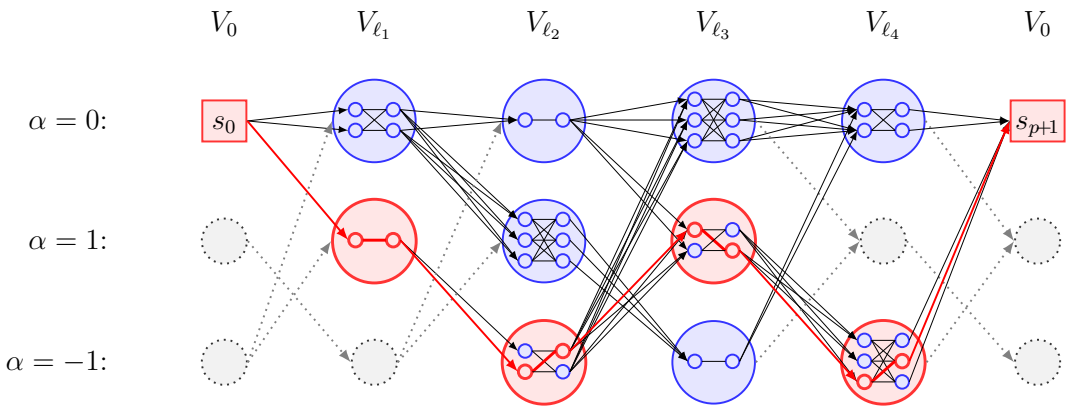

Figure 2.2: Auxiliary graph $\widehat{G_k^*}$ for $k = 2$, current solution $\sigma = (\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5)$ with $\sigma_0 = \sigma_5 = (0, \{0\}, 0)$ and $p = 4$ customer clusters visited in the sequence $V_{\ell_1}, V_{\ell_2}, V_{\ell_3}, V_{\ell_4}$. The neighbor $\sigma' = (\sigma'_0, \sigma'_1, \sigma'_2, \sigma'_3, \sigma'_4, \sigma'_5) \in \mathcal{N}_2^{BS}(\sigma)$ implied by the highlighted s_0-s_{p+1}-path visits the customer clusters in the sequence $V_{\ell_2}, V_{\ell_1}, V_{\ell_4}, V_{\ell_3}$.

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 (2.1).

In Figure 2.2, 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 $\mathcal{O}(p\lambda_{\max}^2 k^2 2^k)$ time and space, where $\lambda_{\max} = \max_{1 \le i \le p} \lambda_{\ell_i}$ 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 $\lambda_{\max}^2$).

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 three 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 *flexible version* allows changing particular entry-exit combinations in the following ways:

Connect When two subroutes $\sigma^1 = (\dots, \sigma_{i-1}^1, \sigma_i^1)$ with $\sigma_i^1 = (e_i^1, V_{\ell_i}^1, f_i^1)$ and $\sigma^2 = (\sigma_j^2, \sigma_{j+1}^2, \dots)$ with $\sigma_j^2 = (e_j^2, V_{\ell_j}^2, f_j^2)$ are concatenated, the exit of cluster σ_i^1 and the entry of cluster σ_j^2 can be modified. We minimize the value $\hat{c}_{e_i^1, f_i'^1} + c_{f_i'^1, e_j'^2} + \hat{c}_{e_j'^2, f_j^2}$ over $(f_i'^1, e_j'^2) \in V_{\ell_i}^1 \times V_{\ell_j}^2$. Note that e_i^1 and f_j^2 are still kept fixed. The resulting subroute is $(\dots, \sigma_{i-1}^1, \sigma_i'^1, \sigma_j'^2, \sigma_{j+1}^2, \dots)$ with $\sigma_i'^1 = (e_i^1, V_{\ell_i}^1, f_i'^1)$ and $\sigma_j'^2 = (e_j'^2, V_{\ell_j}^2, f_j^2)$.

Insert Inserting a cluster $\sigma_a = (e_a, V_{\ell_a}, f_a)$ into route $\sigma = (\dots, \sigma_i, \sigma_j, \dots)$ 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 $\sigma' = (\dots, \sigma_i, \sigma_a', \sigma_j, \dots)$ with $\sigma_a' = (e_a', V_{\ell_a}, f_a')$.

Note that the computational effort for minimization is in both cases bounded by $\mathcal{O}(\lambda_{\max}^2)$, 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 $\sigma^1 = (\sigma_0^1, \sigma_1^1, \dots, \sigma_{i-1}^1, \sigma_i^1, \sigma_{i+1}^1, \dots, \sigma_p^1, \sigma_{p+1}^1)$ and $\sigma^2 = (\sigma_0^2, \sigma_1^2, \dots, \sigma_{j-1}^2, \sigma_j^2, \sigma_{j+1}^2, \dots, \sigma_q^2, \sigma_{q+1}^2)$.

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_i^1 is removed from σ^1 and inserted after σ_j^2 into σ^2 resulting in the two new routes $\sigma'^1 = (\sigma_0^1, \sigma_1^1, \ldots, \sigma_{i-1}'^1, \sigma_{i+1}'^1, \ldots, \sigma_p^1, \sigma_{p+1}^1)$ and $\sigma'^2 = (\sigma_0^2, \sigma_1^2, \ldots, \sigma_{j-1}^2, \sigma_j^2, \sigma_i'^1, \sigma_{j+1}^2, \ldots, \sigma_q^2, \sigma_{q+1}^2)$. In the fixed version, all triplets remain unchanged so that $\sigma_{i-1}'^1 = \sigma_{i-1}^1$, $\sigma_{i+1}'^1 = \sigma_{i+1}^1$, and $\sigma_i'^1 = \sigma_i^1$. In the flexible version, the new routes are derived by applying *Connect* to the subroutes $(\sigma_0^1, \sigma_1^1, \ldots, \sigma_{i-1}^1)$ and $(\sigma_{i+1}^1, \ldots, \sigma_p^1, \sigma_{p+1}^1)$ to produce σ'^1 and by applying *Insert* to the subroutes $(\sigma_0^2, \sigma_1^2, \ldots, \sigma_{j-1}^2, \sigma_j^2)$ and $(\sigma_{j+1}^2, \ldots, \sigma_q^2, \sigma_{q+1}^2)$ and the relocated cluster σ_i^1 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 flexible, 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 flexible 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 flexible version.

Our version of VND uses the nine neighborhoods listed in Table 2.1. In pretests we also analyzed different possible sequences of applying the neighborhoods and different pivoting strategies. Concerning the sequence of neighborhoods, it is common practice to first apply those neighborhoods that can be searched quickly.

Neigborhood	fixed version (no entry-exit modification)	flexible version (with entry-exit modification)
Balas-Simonetti $\mathcal{N}^{BS}_{k_{\text{VND}}}$		1, best improvement, but only once
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 2.1: Priorities $prio(\mathcal{N})$ and pivoting strategy of the nine VND neighborhoods $\mathcal{N}$.

Therefore, we start with the linear (in the route length) neighborhood $\mathcal{N}^{BS}_{k_{\text{VND}}}$ (the choice of a reasonable parameter k_{VND} is analyzed in detail in Section 2.4.2), then apply all four neighborhoods in the fixed version before those using the flexible version. Pretests 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 $\mathcal{N}^{\text{swap}}$. For this reason, the two neighborhoods have identical priorities, see Table 2.1. Finally, a first improvement pivoting strategy was most of the time faster without deteriorating the solution quality for the cluster exchange neighborhoods (note that $\mathcal{N}^{BS}_{k_{\text{VND}}}$ 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 2.1.

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, *priority* is initialized to 1 (see Step 2) and directly incremented (in Step 4) so that the first searched neighborhood is the fixed version of $\mathcal{N}^{\text{1-reloc}}$ (with $prio(\mathcal{N}^{\text{1-reloc}}) = 2$, cf. Table 2.1). Second, for reasonably (small) parameters k_{VND}, the neighborhood $\mathcal{N}^{BS}_{k_{\text{VND}}}$ can be searched quickly. However, we found that almost always a solution once improved with $\mathcal{N}^{BS}_{k_{\text{VND}}}$ cannot be improved with the same neighborhood directly afterwards. It means that no proper local search with $\mathcal{N}^{BS}_{k_{\text{VND}}}$ is necessary. We therefore apply $\mathcal{N}^{BS}_{k_{\text{VND}}}$ only once (deviating from Algorithm 1) and directly continue with the fixed version of $\mathcal{N}^{\text{1-reloc}}$.

2.3.4 LNS Operators

Large neighborhood search (LNS) was originally introduced by Shaw (1998) for the CVRP. A similar idea, called *ruin and recreate*, can be found in (Schrimpf *et al.*,

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  priority := 1
3  repeat
4      priority := priority + 1
5      nb := number of neighborhoods N with prio(N) = priority
6      repeat
7          N := the (iter modulo nb)th neighborhood with prio(N) = priority
8          Search N with pivoting strategy pivot(N) for improving neighbors
9          if improving neighbor x' ∈ N(x) found then
10             x := x'
11             priority := 0
12         iter := iter +1
13     until (priority = 0) or (up to nb times)
14 until priority > max_N prio(N)
```

Output: Local optimum $x = (\sigma^1, \sigma^2, \ldots, \sigma^m)$ w.r.t. all neighborhoods $\{\mathcal{N}\}$

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 (2006a) 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 2.4.1 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 2.3.3. Such a post-optimization of solutions with the help of a local search was also used by Ropke (2009). Finally, the LMNS uses the record-to-record acceptance criterion instead of the Metropolis criterion used by Ropke and Pisinger (2006a,b).

Next, we describe the destroy and repair operators in the remainder of this section and provide a summary of the overall algorithm in Section 2.3.5.

Destroy Operators 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.

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 generate new routes until the required number of m routes is restored.

Both repair operators use the procedure *Insert* as described in Section 2.3.3 to execute the move.

2.3.5 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 2.3.1). 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 auxiliary

Algorithm 2: LMNS algorithm for the CluVRP

Input: Iterations It_{ATSP} and It_{LMNS}
Parameters k_{ATSP}, k_{VND}, and k_{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 ϵ, $\tau_{\min}, \tau_{\max}$, and ρ

Preprocessing(It_{ATSP}, k_{ATSP})
$x := x^{accepted} := x^{best} :=$ Regret-based Savings Algorithm()
for $iter := 1, \ldots, It_{\text{LMNS}}$ **do**
    if x is feasible **then**
        $x :=$ VND(k_{VND}, x)
        $x :=$ Single Improvement with $\mathcal{N}^{BS}_{k_{\text{LMNS}}}(x)$
        if $c(x) < c(x^{best})$ **then**
            $x^{best} := x$
        if AcceptanceCriterion(ϵ, x, x^{best}) **then**
            $x^{accepted} := x$
    Randomly choose $\tau \in \{\tau_{\min}, \ldots, \tau_{\max}\}$
    Randomly choose $\text{Op}^{destroy}$ according to weights $(\psi^{random}, \psi^{related}, \psi^{worst}, \psi^{route})$
    Randomly choose Op^{repair} according to weights $(\omega^{nearest}, \omega^{best})$
    $x := \text{Op}^{repair}(\text{Op}^{destroy}(\tau, \rho, x^{accepted}))$

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. If the savings algorithm constructs a solution with too many routes, we start from a bin-packing solution computed with CPLEX (Valério de Carvalho, 1999).

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 2.3.3. 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 2.4.2 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 operators discussed in Section 2.3.4, creating the starting solution for the next iteration.

2.4 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. These 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-GC` by Bektaş *et al.* (2011) comprises ten small-sized CluVRP instances with 101 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-Bat` was proposed by Battarra *et al.* (2014a) 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.

The metaheuristics of Expósito-Izquierdo *et al.* (2016) and Defryn and Sörensen (2017) were tested on an additional benchmark set, in the following referred to as `Golden-Exp`, based on the `Golden1` to `Golden20` instances. Expósito-Izquierdo *et al.* (2016) vary the original `Golden` instances by generating different clusters: Now clusters are generated by setting a maximum filling percentage β relative to the vehicle capacity (see Expósito-Izquierdo *et al.* (2016) for details). Each original instance is altered using five different β-values. We compare the LMNS against the two metaheuristics also on this set of 100 instances.

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 $100\,(z - \text{BKS})/\text{BKS}$. 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.

To statistically support several statements, we use various non-parametric statistical tests, following the tutorial by Derrac *et al.* (2011). In particular, we apply the *Friedman* test (Friedman, 1937) using its extension by Iman and Davenport (1980), the *Finner* procedure (Finner, 1993), and *Shaffer's* static procedure (Shaffer, 1986) for comparisons between several algorithms/parameter settings. We used the `Java` packages `CONTROLTEST` and `MULTIPLETEST` developed by García *et al.* (2010) available on the *Statistical Inference in Computational Intelligence*

and Data Mining website `http://sci2s.ugr.es/sicidm`. Herewith, we calculate *p-values* and *adjusted p-values* (APVs) resulting from the above tests. To be precise, in the following, all p-values in Friedman's test are computed according to the Iman-Davenport extension, and we use $\alpha = 0.05$ as the level of significance, if not stated differently.

2.4.1 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. All parameter studies were conducted with the `GVRP-GC`, `Golden-Bat`, and `Li` instances, i.e., we excluded the benchmark `Golden-Exp` in order to not use identical customer sets too often.

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 $\lambda_{\text{BS}} = 10$. 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 $\lambda_{\text{BS}} = 10$, 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 $k_{\text{ATSP}} = 3$ and $It_{\text{ATSP}} = 50$.

We summarize what criteria we studied to find the combination $k_{\text{ATSP}} = 3$ and $It_{\text{ATSP}} = 50$. Over the `GVRP-GC`, `Golden-Bat`, and `Li` 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 $k_{\text{ATSP}} = 3$ and $It_{\text{ATSP}} = 50$. 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 2.4.4). However, all entry-exit combinations are correct. If instead the exact SHP solution of Battarra *et al.* (2014a) 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.

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 pretests. Pretests 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 choose $(\psi^{random}, \psi^{related}, \psi^{worst}, \psi^{route}) = (0.3, 0.3, 0.3, 0.1)$ for the weights.

Second, we test the usefulness of each and every operator: Using setting $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ with $It_{\text{LMNS}} = 5\,000$ (see the next section for a study on reasonable $(k_{\text{VND}}, k_{\text{LMNS}})$ combinations), we find that using only one destroy and one repair operator (eight possible setups) is clearly outperformed by the combination of all operators (called *All Operators*): The average *Gap Best* is 0.026 %, while averages range from 0.057 % to 0.942 % if only one destroy and one repair operator is used. The null hypothesis of the Friedman test is that all medians between all settings are equal. We obtain a p-value smaller than 0.001 clearly rejecting the null hypothesis, and the setting *All Operators* is ranked as the best setting by the Friedman test. Hence, we are allowed to apply the Finner procedure, which now shows that the performance of the setting *All Operators* significantly differs from the five settings *Worst-Nearest*, *Worst-Best*, *Route-Nearest*, *Route-Best*, and *Random-Best* (APVs are smaller than 0.01, see Table 2.20 in Appendix 2.C). The differences to the remaining three settings are however not significant.

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	0.040	0.047	0.026
# BKS	213	208	214	213	205	201	217

Table 2.2: Comparison of LMNS using different destroy and repair operators.

Accordingly, Table 2.2 shows the results of disabling a single operator in comparison to the version that uses all six operators. Again, these tests are performed with $It_{\text{LMNS}} = 5\,000$ and the combination $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$. 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 an LMNS that computes 217 best known solutions and at the same time the lowest overall *Gap Best* of 0.026 %. The six other variants compute between 201 and 214 *best known solutions* (BKS) with an average *Gap Best* of at least 0.032 %. Again, when comparing for *Gap Best*, the setting *All Operators* is ranked as the best setting in Friedman's test, however, the settings cannot be shown to differ significantly (the p-value is 0.42). If we do not consider those instances where each and every setting finds the BKS, leading to 52 remaining CluVRP instances, some results become significant: The p-value in Friedman's test is smaller than 0.001, and the setting *All Operators* significantly differs from the two settings that use only one repair operator (Finner APVs are smaller than 0.001, see Table 2.21 in Appendix 2.C). However, differences to the settings that use only one destroy operator are not significant.

Even if not always statistically significant, it seems that all operators contribute to the quality of LMNS. Hence, we use all six operators with the weights given above for the remaining experiments.

2.4.2 Usefulness of the Balas-Simonetti Neighborhood

In this section, we analyze the generalized version of the Balas-Simonetti neighborhood (see Section 2.3.2) 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 $k_{\text{LMNS}} = 0$,

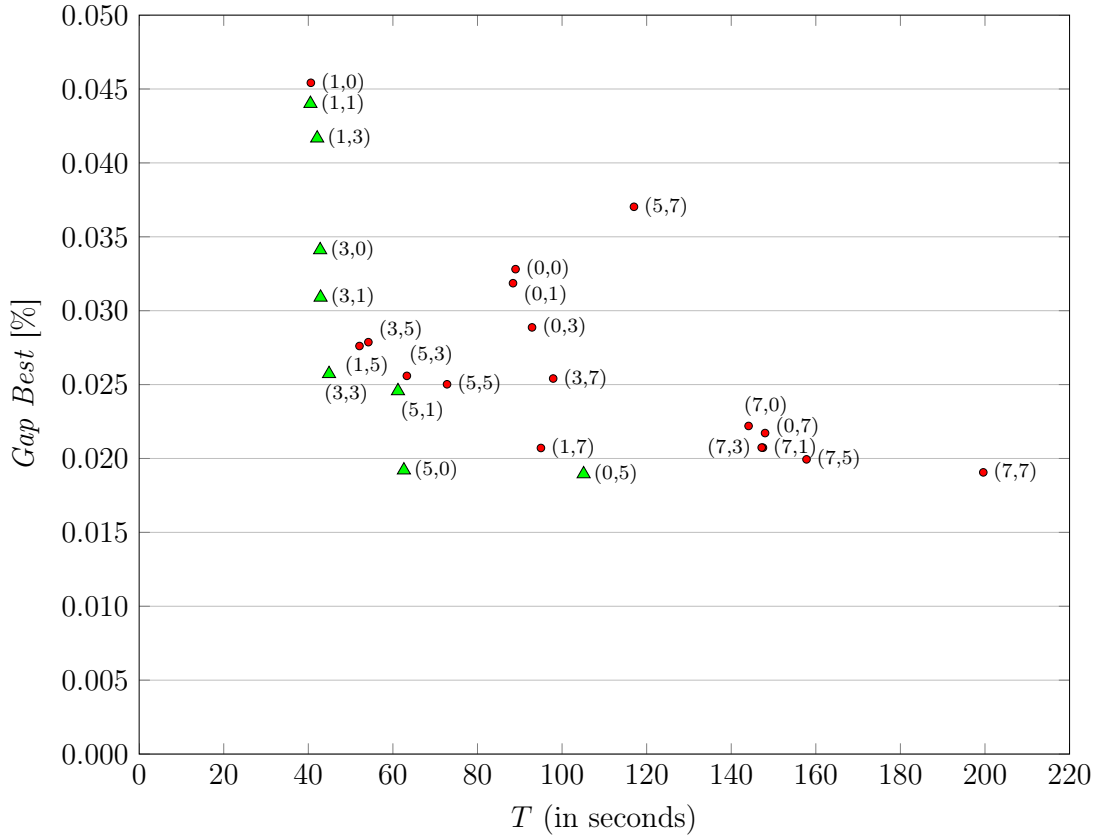

Figure 2.3: Comparison of LMNS with different combinations $(k_{\text{VND}}, k_{\text{LMNS}})$. Green triangles $\triangle$ indicate Pareto-optimal combinations, dominated combinations are indicated as red circles $\circ$.

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 2.3 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 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-GC 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 better results.

We are not aware of any statistical tests that can reasonably consider the tradeoff between computation time and solution quality. Hence, we analyze both criteria (time and quality) separately.

Friedman's test for the computation times T confirms that there are significant differences (p-value is smaller than 0.001). Using Shaffer's procedure (for multiple comparisons without control method), differences between the different $(k_{\text{VND}}, k_{\text{LMNS}})$ combinations are most of the time significant (255 of the 300 APVs are smaller than 0.05). Moreover, the ranking according to Friedman's test is similar to the ordering given by the runtimes T (average over all 242 instances) displayed in Figure 2.3.

On the contrary, applying Friedman's test for *Gap Best* obtained with different parameters $(k_{\text{VND}}, k_{\text{LMNS}})$, the four settings with $(k_{\text{VND}}, k_{\text{LMNS}}) = (0, 1), (1, 1), (1, 0), (0, 0)$, which do not use the Balas-Simonetti neighborhood at all or only for changing the entrance-exit decisions, are ranked worst. Moreover, settings with higher values of k_{VND} as well as k_{LMNS} tend to be ranked better. These results are however not significant because the p-value is 0.89. Again, when we consider only the 48 instances where not all settings find the BKS, results become significant: According to Shaffer's procedure, the four above mentioned settings are the only settings whose *Gap Best* value differs significantly from some of the other settings (using the significance level $\alpha = 0.1$).

The reader may wonder why the statistical tests cannot confirm the rather clear message about solution quality that Figure 2.3 gives. First, differences between some Pareto-optimal combination and dominated combinations are reasonably small, in particular w.r.t. *Gap Best*. Second, in Figure 2.3 the combinations result from averages (over T and *Gap Best*), while the statistical test use ranks. Hence, a small positive and a huge negative difference lead to identical ranks while they cause visible differences in the average values.

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.

2.4.3 Comparison to Results of Expósito-Izquierdo *et al.* (2016) and Defryn and Sörensen (2017)

For the comparison with the metaheuristics of Expósito-Izquierdo *et al.* (2016) and Defryn and Sörensen (2017), we use the parameter combinations $(k_{\text{VND}}, k_{\text{LMNS}}) =$

$(3, 3)$ and $(5, 0)$ with 5 000 and 50 iterations, respectively. With these settings, our computation times become comparable with the two metaheuristics that primarily focus on speed.

For the `Golden-Exp` benchmark, both papers allow a computation time of 60 seconds, what our LMNS does not exceed with 5 000 iterations. In Table 2.3, we present results for the combination $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$. Entry T is the total computation time in seconds and T_p the time spent with preprocessing. Overall, we find 75 out of 100 BKS with an average *Gap Best* of 0.05 % compared to 1.76 % and 0.68 % (with 35 and 15 BKS) by Expósito-Izquierdo *et al.* (2016) and Defryn and Sörensen (2017), respectively. Our average runtime of 21 seconds is smaller compared to the time limit of 60 seconds chosen in the two other papers. Comparing the LMNS with both metaheuristics, results for *Gap Best* and T are confirmed to be highly significant (Friedman's test gives p-values smaller than 0.001 in both cases, all APVs of Shaffer's procedure (which pairwise compare the LMNS settings with one of the other methods) are smaller than 0.001, see Tables 2.22 and 2.23 in Appendix 2.C).

	Expos.-Iz. *et al.* (2016)				Defr. and Sör. (2017)				LMNS (3, 3, 5 000)				
β	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T_p	T	*Gap Best*	*Gap Avg.*	# BKS
0.1	60	3.86	n.a.	0	60	1.56	n.a.	1	0.1	25	0.04	0.53	13
0.25	60	2.77	n.a.	0	60	0.92	n.a.	0	0.5	13	0.06	0.27	16
0.5	60	2.05	n.a.	0	60	0.56	n.a.	3	4.0	15	0.10	0.11	18
0.75	60	0.12	n.a.	15	60	0.16	n.a.	8	14.7	17	<0.01	<0.01	17
1.0	60	0.00	n.a.	20	60	0.20	n.a.	3	33.6	35	0.03	0.03	11
Total	60	1.76	n.a.	35	60	0.68	n.a.	15	10.6	21	0.05	0.19	75

Table 2.3: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ with 5 000 iterations and benchmark set `Golden-Exp` (20 instances per filling percentage β) with the number n of customers ranging from 200 to 483.

Note that for the `Golden-Bat` instances both methods of Expósito-Izquierdo *et al.* (2016) and Defryn and Sörensen (2017) focus on producing reasonable solutions within ten seconds, while our method already consumes a remarkable portion of time just for the preprocessing. Limiting the number of iterations of our LMNS to 50 is certainly in contrast to the general design principles of an LNS; we do this only in order to match the short time limit on average. Summarizing Table 2.4, we find 114 out of 220 BKS with an average *Gap Best* of 0.18 % compared to 3.00 % and 1.12 % (with an unknown number and 8 BKS) by Expósito-Izquierdo *et al.* (2016) and Defryn and Sörensen (2017). Comparing the LMNS with the method of Defryn and Sörensen (2017), Friedman's test confirms significant differences (the

p-values are smaller than 0.01) and ranks LMNS as the better method for *Gap Best*. Interestingly, LMNS is also identified as significantly faster, although the average time T clearly exceeds the average time achieved by Defryn and Sörensen (2017). Such a result is possible because the statistical analyses are based on ranks and not on averages; see Tables 2.24 and 2.25 in Appendix 2.C for the APVs of Shaffer's procedure. Note also that in (Expósito-Izquierdo *et al.*, 2016) only average results are reported so that we cannot further analyze the differences with statistical tests.

		Expos.-Iz. *et al.* (2016)				Defr. and Sör. (2017)				LMNS $(3, 3, 50)$				
n	#	T	*Gap Best*	*Gap Avg.*	# BKS	T	*Gap Best*	*Gap Avg.*	# BKS	T_p	T	*Gap Best*	*Gap Avg.*	# BKS
200	11	10	4.61	n.a.	n.a.	4.5	0.07	0.29	6	9.8	9.9	0.10	0.53	8
240	22	10	2.39	n.a.	n.a.	4.1	0.44	0.86	1	3.5	3.7	0.05	0.31	17
252	11	10	0.50	n.a.	n.a.	4.0	0.53	0.88	0	1.2	1.4	0.10	0.29	8
255	11	10	3.69	n.a.	n.a.	3.9	1.33	2.08	0	2.0	2.1	0.09	0.57	8
280	11	10	2.94	n.a.	n.a.	4.6	0.71	1.14	0	19.8	20.0	0.05	0.37	7
300	11	10	1.04	n.a.	n.a.	4.0	0.93	1.38	0	6.0	6.3	0.06	0.22	8
320	22	10	1.26	n.a.	n.a.	4.5	0.85	1.31	0	4.7	4.9	0.10	0.42	13
323	11	10	4.94	n.a.	n.a.	3.9	0.93	1.61	1	2.4	2.6	0.26	1.08	6
360	22	10	2.87	n.a.	n.a.	4.5	1.02	1.51	0	17.6	17.9	0.09	0.35	14
396	11	10	1.54	n.a.	n.a.	4.0	1.37	1.85	0	2.0	2.4	0.41	0.80	1
399	11	10	4.96	n.a.	n.a.	4.3	2.15	2.86	0	2.4	2.8	0.32	0.86	4
400	11	10	2.56	n.a.	n.a.	4.8	1.26	1.71	0	19.0	19.5	0.15	0.53	3
420	11	10	2.60	n.a.	n.a.	4.2	1.20	1.71	0	15.0	15.4	0.12	0.44	8
440	11	10	3.67	n.a.	n.a.	5.0	1.32	1.83	0	19.4	19.9	0.21	0.59	2
480	22	10	3.42	n.a.	n.a.	4.3	1.49	1.98	0	15.0	15.6	0.33	0.66	6
483	11	10	4.93	n.a.	n.a.	4.1	2.97	3.11	0	2.4	2.9	0.66	1.41	1
Total	220	10	3.00	n.a.	n.a.	4.3	1.12	1.59	8	9.1	9.5	0.18	0.56	114

Table 2.4: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ with 50 iterations and benchmark set `Golden-Bat`, sorted by the number number n of customers (220 instances with an average number θ of nodes per cluster between 5 and 15).

Very similar results are also obtained with the setting $(k_{\text{VND}}, k_{\text{LMNS}}) = (5, 0)$ (we omit their presentation in Tables 2.3 and 2.4). With both combinations $(3, 3)$ and $(5, 0)$, we find 57 new BKS for the `Golden-Exp` benchmark. As our results are more than competitive, we omit further comparisons of the LMNS with the algorithms of Expósito-Izquierdo *et al.* (2016) and Defryn and Sörensen (2017).

2.4.4 Comparison to Results of Vidal *et al.* (2015)

This section provides a comparison of the LMNS and the UHGS approach by Vidal *et al.* (2015). The tables in Vidal *et al.* (2015) include some inconsistencies. In the following, we compare with the true values computed with the UHGS, published in the corrigendum by Vidal *et al.* (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 2.5 and 2.6 summarize aggregated results grouped by setting and instance set. Note that the time T_p spent with preprocessing 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-GC`	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-Bat`	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	0.17	3	314.7	660	0.02	0.17	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 2.5: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ and the benchmark sets `GVRP-GC` (10 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), `Golden-Bat` (220 instances with n between 200 to 483 and θ between 5 and 15), and `Li` (12 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-GC`	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-Bat`	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	0.38	1	3137	0.07	0.19	4	4836	0.06	0.17	4	314.7	660	0.02	0.17	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 2.6: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (5, 0)$ and the same benchmark sets as in Table 2.5.

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-GC` 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-Bat` 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 Table 2.10 of Appendix 2.A).

The `Golden-Bat` 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 2.7 and 2.8. Here, the computation times for the preprocessing T_p are strongly increasing with the average size of the clusters.

	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	<0.01	0.04	19	926	<0.01	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	<0.01	0.02	19	288	<0.01	0.01	19	577	<0.01	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 2.7: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (3, 3)$ and the `Golden-Bat` instances grouped by average cluster size; each group comprises 20 instances.

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 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 a value smaller than 0.01 % (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.

Concerning the statistical significance of the comparison of LMNS and UHGS, we can summarize the following: (i) With respect to *Gap Best*, UHGS is ranked as the best method by Friedman's test, but differences are not significant (p-value = 0.63). However, if we consider only instances where some settings do not find an optimal solution, $\text{LMNS}^{3,3}_{5\,000}$ significantly differs from UHGS and all LMNS settings except $\text{LMNS}^{5,0}_{5\,000}$ (Friedman test with p-value smaller than 0.001; APVs

	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	<0.01	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	<0.01	0.02	19	446	<0.01	0.01	19	665	<0.01	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 2.8: Aggregated results for $(k_{\text{VND}}, k_{\text{LMNS}}) = (5, 0)$ and the `Golden-Bat` instances grouped by average cluster size; each group comprises 20 instances.

of Shaffer's procedure smaller than 0.05), and $\text{LMNS}^{5,0}_{5\,000}$ significantly differs from UHGS, $\text{LMNS}^{3,3}_{100\,000}$, and $\text{LMNS}^{5,0}_{75\,000}$ (p-value and APVs of Shaffer's procedure as before). (ii) With respect to *Gap Avg.*, LMNS is significantly better compared to UHGS (p-value and APVs of Shaffer's procedure as before), except for both settings with 5 000 iterations and setting $\text{LMNS}^{3,3}_{50\,000}$. (iii) UHGS is significantly faster than $\text{LMNS}^{3,3}_{100\,000}$, $\text{LMNS}^{5,0}_{75\,000}$, and $\text{LMNS}^{5,0}_{100\,000}$, but significantly slower than $\text{LMNS}^{3,3}_{50\,000}$, $\text{LMNS}^{3,3}_{5\,000}$, and $\text{LMNS}^{5,0}_{5\,000}$ (p-value and APVs of Shaffer's procedure as before).

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. Last, computation times of $\text{LMNS}^{5,0}_{75\,000}$ and $\text{LMNS}^{5,0}_{100\,000}$ grow much faster than those of UHGS for large-scale instances such as those from the `Li` benchmark.

2.5 Conclusions

In this chapter, we proposed a new metaheuristic for the CluVRP. Our new LMNS approach can be classified as an 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.* (2014a) 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 the 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, one new best solution for the `GVRP-GC` benchmark set, two for the `Li` benchmark, and 57 for the `Golden-Exp` benchmark were found with the LMNS.

Acknowledgment

This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG) under grant no. IR 122/7-1.

Bibliography

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., Erdoğan, G., and Vigo, D. (2014a). 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. (2017). A fast two-level variable neighborhood search for the clustered vehicle routing problem. *Computers & Operations Research*, **83**, 78 – 94.

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, **1**, 3–18.

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.

Finner, H. (1993). On a monotonicity problem in step-down multiple test procedures. *Journal of the American Statistical Association*, **88**(423), 920–923.

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.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, **32**(200), 675–701.

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.

García, S., Fernández, A., Luengo, J., and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, **180**(10), 2044 – 2064. Special Issue on Intelligent Distributed Information Systems.

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

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

Iman, R. L. and Davenport, J. M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in Statistics - Theory and Methods*, **9**(6), 571–595.

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 Optimization. Feature 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.

Shaffer, J. P. (1986). Modified sequentially rejective multiple test procedures. *Journal of the American Statistical Association*, **81**(395), 826–831.

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.

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

Vidal, T. (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.

Vidal, T., Battarra, M., Subramanian, A., and Erdoğan, G. (2017). Corrigendum to "Hybrid metaheuristics for the clustered vehicle routing problem [Comput. Oper. Res., 58 (2015): 87-99]". *Computers & Operations Research*, **85**(Supplement C), 206–207.

Appendix

2.A Detailed Results A

Tables 2.9–2.14 provide detailed results for each instance of sets `GVRP-GC`, `Golden-Bat` and `Li`. 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.* (2014a)	642	642	0.1	83
C	120	61	4	807	Battarra *et al.* (2014a)	807	807	0.1	215
C	150	76	6	816	Battarra *et al.* (2014a)	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.* (2014a)	607	607	0.1	100
C	120	41	3	691	Battarra *et al.* (2014a)	691	691	0.1	158
C	150	51	4	804	Battarra *et al.* (2014a)	804	804	0.1	181
C	199	67	6	908	Battarra *et al.* (2014a)	908	908	0.1	347

Table 2.9: Detailed results for `GVRP-GC` 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	41364	LMNS*	41431	41485.9	5.2	10946

Table 2.10: Detailed results for `Li` instances; * found with LMNS during experimentation.

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

Table 2.11: Detailed results for the Golden-Bat 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.* (2014a)	8624	8624	58.0	228
Golden6	280	21	3	8628	Battarra *et al.* (2014a)	8628	8628	48.6	229
Golden6	280	22	3	8646	Battarra *et al.* (2014a)	8646	8646	29.2	223
Golden6	280	24	4	8853	Battarra *et al.* (2014a)	8853	8853	18.0	251
Golden6	280	26	4	8910	Battarra *et al.* (2014a)	8910	8910	18.0	270
Golden6	280	29	4	8936	Battarra *et al.* (2014a)	8936	8936	5.7	358
Golden6	280	32	4	8891	Battarra *et al.* (2014a)	8891	8891	3.8	381
Golden6	280	36	4	8969	Battarra *et al.* (2014a)	8969	8969	3.7	374
Golden6	280	41	4	9028	Battarra *et al.* (2014a)	9028	9028	3.8	423
Golden6	280	47	4	8923	Battarra *et al.* (2014a)	8923	8923	3.6	495
Golden6	280	57	4	9028	Battarra *et al.* (2014a)	9028	9028	1.0	644
Golden7	360	25	3	9904	Battarra *et al.* (2014a)	9904	9904	56.8	409
Golden7	360	26	3	9888	Battarra *et al.* (2014a)	9888	9888	39.8	413
Golden7	360	28	3	9917	Battarra *et al.* (2014a)	9917	9917	38.4	409
Golden7	360	31	4	10021	Battarra *et al.* (2014a)	10021	10021	28.5	492
Golden7	360	33	4	10029	Battarra *et al.* (2014a)	10029	10029	21.3	508
Golden7	360	37	4	10131	Battarra *et al.* (2014a)	10131	10131	20.7	551
Golden7	360	41	4	10052	Battarra *et al.* (2014a)	10052	10052	20.7	661
Golden7	360	46	4	10080	Battarra *et al.* (2014a)	10080	10080	6.8	701
Golden7	360	52	4	10095	Battarra *et al.* (2014a)	10095	10095	1.2	768
Golden7	360	61	4	10096	Battarra *et al.* (2014a)	10096	10096	1.1	858
Golden7	360	73	4	10014	Battarra *et al.* (2014a)	10014	10014	1.2	1139
Golden8	440	30	4	10866	Battarra *et al.* (2014a)	10866	10866	31.0	599
Golden8	440	32	4	10831	Battarra *et al.* (2014a)	10831	10831	31.2	638
Golden8	440	34	4	10847	Battarra *et al.* (2014a)	10847	10847	31.1	664
Golden8	440	37	4	10859	Battarra *et al.* (2014a)	10859	10859	27.4	678
Golden8	440	41	4	10934	Battarra *et al.* (2014a)	10934	10934	27.0	700
Golden8	440	45	4	10960	Battarra *et al.* (2014a)	10960	10960	26.4	769
Golden8	440	49	4	11042	Battarra *et al.* (2014a)	11042	11042	5.9	826
Golden8	440	56	4	11194	Battarra *et al.* (2014a)	11194	11194.3	2.9	997
Golden8	440	63	4	11252	Battarra *et al.* (2014a)	11252	11252	2.8	993
Golden8	440	74	4	11321	Battarra *et al.* (2014a)	11321	11321	2.8	1249
Golden8	440	89	4	11209	Battarra *et al.* (2014a)	11209	11209.4	1.5	1652
Golden9	255	18	4	300	Battarra *et al.* (2014a)	300	300	4.4	185
Golden9	255	19	4	299	Battarra *et al.* (2014a)	299	299	3.9	171
Golden9	255	20	4	296	Battarra *et al.* (2014a)	296	296	3.2	203
Golden9	255	22	4	290	Battarra *et al.* (2014a)	290	290	2.4	210
Golden9	255	24	4	290	Battarra *et al.* (2014a)	290	290	1.9	222
Golden9	255	26	4	288	Battarra *et al.* (2014a)	288	288	1.3	232
Golden9	255	29	4	292	Battarra *et al.* (2014a)	292	292	0.8	256
Golden9	255	32	4	297	Battarra *et al.* (2014a)	297	297	0.7	272
Golden9	255	37	4	294	Battarra *et al.* (2014a)	294	294	0.6	317
Golden9	255	43	4	295	Battarra *et al.* (2014a)	295	295.6	0.4	367
Golden9	255	52	4	296	Battarra *et al.* (2014a)	296	296.9	0.1	432
Golden10	323	22	4	367	Battarra *et al.* (2014a)	367	367	6.6	234
Golden10	323	24	4	361	Battarra *et al.* (2014a)	361	361	3.7	232
Golden10	323	25	4	359	Battarra *et al.* (2014a)	359	360.2	3.2	249
Golden10	323	27	4	361	Battarra *et al.* (2014a)	361	361	3.0	274
Golden10	323	30	4	367	Battarra *et al.* (2014a)	368	368	2.5	290
Golden10	323	33	4	373	Battarra *et al.* (2014a)	375	375	1.8	302
Golden10	323	36	4	385	Battarra *et al.* (2014a)	385	385.3	1.2	314
Golden10	323	41	4	400	Battarra *et al.* (2014a)	400	400	0.5	330
Golden10	323	47	4	398	Battarra *et al.* (2014a)	398	398	0.5	366
Golden10	323	54	4	393	Battarra *et al.* (2014a)	393	393.4	0.4	418
Golden10	323	65	4	387	Battarra *et al.* (2014a)	387	387.6	0.3	504

Table 2.12: Detailed results for the Golden-Bat 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.* (2014a)	457	457	5.4	452
Golden11	399	29	5	455	Battarra *et al.* (2014a)	455	455	5.0	472
Golden11	399	31	5	455	Battarra *et al.* (2014a)	455	455	4.2	467
Golden11	399	34	5	455	Battarra *et al.* (2014a)	455	455	3.0	547
Golden11	399	37	5	459	Battarra *et al.* (2014a)	459	459	2.4	553
Golden11	399	40	5	461	Battarra *et al.* (2014a)	461	461	1.3	586
Golden11	399	45	5	462	Battarra *et al.* (2014a)	462	462	1.1	658
Golden11	399	50	5	458	Battarra *et al.* (2014a)	458	458	1.0	674
Golden11	399	58	5	456	Battarra *et al.* (2014a)	456	456	0.8	734
Golden11	399	67	5	454	Battarra *et al.* (2014a)	454	454	0.5	896
Golden11	399	80	5	451	Battarra *et al.* (2014a)	451	451	0.2	1028
Golden12	483	33	5	535	Battarra *et al.* (2014a)	535	535	5.4	595
Golden12	483	35	5	537	Battarra *et al.* (2014a)	537	538.2	5.0	619
Golden12	483	38	5	535	Battarra *et al.* (2014a)	535	538.6	3.7	652
Golden12	483	41	5	537	Battarra *et al.* (2014a)	537	537	2.5	682
Golden12	483	44	5	535	Battarra *et al.* (2014a)	535	536.4	2.4	715
Golden12	483	49	5	533	Battarra *et al.* (2014a)	533	533.4	2.3	810
Golden12	483	54	5	535	Battarra *et al.* (2014a)	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.* (2014a)	552	552	3.2	158
Golden13	252	19	4	549	Battarra *et al.* (2014a)	549	549	2.1	186
Golden13	252	20	4	548	Battarra *et al.* (2014a)	548	548	2.0	209
Golden13	252	22	4	548	Battarra *et al.* (2014a)	548	548	1.7	224
Golden13	252	23	4	548	Battarra *et al.* (2014a)	548	548	1.6	231
Golden13	252	26	4	542	Battarra *et al.* (2014a)	542	542	1.1	250
Golden13	252	29	4	540	Battarra *et al.* (2014a)	540	540	0.7	292
Golden13	252	32	4	543	Battarra *et al.* (2014a)	543	543	0.6	289
Golden13	252	37	4	545	Battarra *et al.* (2014a)	545	545	0.3	297
Golden13	252	43	4	553	Battarra *et al.* (2014a)	553	553	0.1	381
Golden13	252	51	4	560	Battarra *et al.* (2014a)	560	560	0.1	449
Golden14	320	22	4	692	Battarra *et al.* (2014a)	692	692	4.9	267
Golden14	320	23	4	688	Battarra *et al.* (2014a)	688	688	3.7	266
Golden14	320	25	4	678	Battarra *et al.* (2014a)	678	678	2.9	255
Golden14	320	27	4	676	Battarra *et al.* (2014a)	676	676	2.3	275
Golden14	320	30	4	678	Battarra *et al.* (2014a)	678	678	1.7	313
Golden14	320	33	4	682	Battarra *et al.* (2014a)	682	682	1.6	349
Golden14	320	36	4	687	Battarra *et al.* (2014a)	687	687	0.8	345
Golden14	320	41	4	690	Battarra *et al.* (2014a)	690	690	0.5	414
Golden14	320	46	4	694	Battarra *et al.* (2014a)	694	694	0.3	453
Golden14	320	54	4	699	Battarra *et al.* (2014a)	699	699	0.1	509
Golden14	320	65	4	703	Battarra *et al.* (2014a)	703	703	0.1	655
Golden15	396	27	4	842	Battarra *et al.* (2014a)	842	842	5.3	359
Golden15	396	29	4	843	Battarra *et al.* (2014a)	843	843	4.5	385
Golden15	396	31	4	837	Battarra *et al.* (2014a)	837	837	3.3	379
Golden15	396	34	4	838	Battarra *et al.* (2014a)	838	838	2.3	404
Golden15	396	37	4	845	Battarra *et al.* (2014a)	845	845	1.8	397
Golden15	396	40	4	849	Battarra *et al.* (2014a)	849	849	1.3	465
Golden15	396	45	5	853	Battarra *et al.* (2014a)	853	853	0.9	518
Golden15	396	50	5	851	Battarra *et al.* (2014a)	851	851	0.7	562
Golden15	396	57	5	850	Battarra *et al.* (2014a)	850	850	0.5	638
Golden15	396	67	5	855	Battarra *et al.* (2014a)	855	855.6	0.1	688
Golden15	396	80	5	857	Battarra *et al.* (2014a)	857	857.6	0.1	920

Table 2.13: Detailed results for the Golden-Bat 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.* (2014a)	1030	1030	6.7	656
Golden16	480	35	5	1028	Battarra *et al.* (2014a)	1028	1028	5.0	661
Golden16	480	37	5	1028	Battarra *et al.* (2014a)	1028	1028	3.7	704
Golden16	480	41	5	1032	Battarra *et al.* (2014a)	1032	1032	2.6	759
Golden16	480	44	5	1028	Battarra *et al.* (2014a)	1028	1028	2.0	756
Golden16	480	49	5	1031	Battarra *et al.* (2014a)	1031	1031	1.4	822
Golden16	480	54	5	1022	Battarra *et al.* (2014a)	1022	1022	1.3	914
Golden16	480	61	5	1013	Battarra *et al.* (2014a)	1014	1014	1.0	1046
Golden16	480	69	5	1012	Battarra *et al.* (2014a)	1012	1012	0.8	1203
Golden16	480	81	5	1018	Battarra *et al.* (2014a)	1018	1018	0.3	1376
Golden16	480	97	5	1018	Battarra *et al.* (2014a)	1019	1019.6	0.1	1594
Golden17	240	17	3	418	Battarra *et al.* (2014a)	418	418	6.9	196
Golden17	240	18	3	419	Battarra *et al.* (2014a)	419	419	5.9	215
Golden17	240	19	3	422	Battarra *et al.* (2014a)	422	422	4.8	213
Golden17	240	21	3	425	Battarra *et al.* (2014a)	425	425	4.1	220
Golden17	240	22	3	424	Battarra *et al.* (2014a)	424	424	3.9	215
Golden17	240	25	3	418	Battarra *et al.* (2014a)	418	418	1.8	250
Golden17	240	27	3	414	Battarra *et al.* (2014a)	414	414	1.3	250
Golden17	240	31	4	421	Battarra *et al.* (2014a)	421	421	0.5	308
Golden17	240	35	4	417	Battarra *et al.* (2014a)	417	417	0.2	322
Golden17	240	41	4	412	Battarra *et al.* (2014a)	412	412	0.1	400
Golden17	240	49	4	414	Battarra *et al.* (2014a)	414	414	0.1	473
Golden18	300	21	4	592	Battarra *et al.* (2014a)	592	592	14.7	231
Golden18	300	22	4	594	Battarra *et al.* (2014a)	594	594	14.1	245
Golden18	300	24	4	592	Battarra *et al.* (2014a)	592	592	14.4	253
Golden18	300	26	4	590	Battarra *et al.* (2014a)	590	590	6.4	303
Golden18	300	28	4	577	Battarra *et al.* (2014a)	577	577	3.1	364
Golden18	300	31	4	578	Battarra *et al.* (2014a)	578	578	2.2	365
Golden18	300	34	4	582	Battarra *et al.* (2014a)	582	582	1.5	383
Golden18	300	38	4	586	Battarra *et al.* (2014a)	586	586	1.1	407
Golden18	300	43	4	594	Battarra *et al.* (2014a)	594	594	0.5	427
Golden18	300	51	4	601	Battarra *et al.* (2014a)	601	601	0.1	521
Golden18	300	61	4	599	Battarra *et al.* (2014a)	599	599	0.1	659
Golden19	360	25	10	925	Battarra *et al.* (2014a)	925	925	35.6	352
Golden19	360	26	10	924	Battarra *et al.* (2014a)	924	924	29.4	365
Golden19	360	28	4	808	Battarra *et al.* (2014a)	808	808	21.2	327
Golden19	360	31	4	811	Battarra *et al.* (2014a)	812	812	9.9	391
Golden19	360	33	4	797	Battarra *et al.* (2014a)	797	797	4.5	426
Golden19	360	37	5	799	Battarra *et al.* (2014a)	799	799	1.9	490
Golden19	360	41	5	789	Battarra *et al.* (2014a)	789	789	1.1	592
Golden19	360	46	5	788	Battarra *et al.* (2014a)	788	788	1.0	625
Golden19	360	52	5	800	Battarra *et al.* (2014a)	800	800	0.9	691
Golden19	360	61	5	807	Battarra *et al.* (2014a)	807	807	0.6	769
Golden19	360	73	5	810	Battarra *et al.* (2014a)	810	810	0.4	953
Golden20	420	29	11	1220	Battarra *et al.* (2014a)	1220	1220	43.1	477
Golden20	420	31	12	1232	Battarra *et al.* (2014a)	1232	1232	28.3	488
Golden20	420	33	12	1208	Battarra *et al.* (2014a)	1208	1208	25.9	514
Golden20	420	36	5	1059	Battarra *et al.* (2014a)	1059	1059	15.0	485
Golden20	420	39	5	1052	Battarra *et al.* (2014a)	1052	1052	7.8	523
Golden20	420	43	5	1052	Battarra *et al.* (2014a)	1052	1052	4.5	543
Golden20	420	47	5	1053	Battarra *et al.* (2014a)	1053	1053	4.5	646
Golden20	420	53	5	1058	Battarra *et al.* (2014a)	1058	1058	4.4	689
Golden20	420	61	5	1058	Battarra *et al.* (2014a)	1058	1058	4.3	828
Golden20	420	71	5	1049	Battarra *et al.* (2014a)	1059	1059	3.7	984
Golden20	420	85	5	1049	Battarra *et al.* (2014a)	1049	1049	0.7	1119

Table 2.14: Detailed results for the Golden-Bat instances 16-20.

2.B Detailed Results B

Tables 2.15–2.19 provide detailed results for each instance of set `Golden-Exp`. Here, all LMNS runs were performed with the setting $(k_{\text{VND}}, k_{\text{LMNS}}, It_{\text{LMNS}}) = (3, 3, 5\,000)$.

Expósito-Izquierdo *et al.* (2016) considered Euclidean distances for these instances. For our approach and these instances, Euclidean distances are rounded to the third decimal place. No rounding rule is given in (Expósito-Izquierdo *et al.*, 2016), so that we can bound the maximum difference in computed objective values to $0.0005 \times n$. Accordingly, we consider our solutions as new BKS only if the computed objective value improves the BKS by more than $0.0005 \times n$. As a result, value *Best* and *Avg.* for LMNS in Tables 2.15–2.19 might be slightly better than the previously known BKS. However, we do not consider them as new BKS.

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
1	240	121	9	5620.17	LMNS	5620.17	5646.83	0.1	28
2	320	102	10	9091.61	LMNS	9091.61	9152.21	0.1	47
3	400	97	9	12632.80	LMNS	12632.80	12766.10	0.1	14
4	480	105	10	17263.80	LMNS	17263.80	17347.40	0.1	58
5	200	50	5	8909.06	LMNS	8909.06	8909.06	0.1	16
6	280	68	7	10775.10	LMNS	10778.60	10809.00	0.1	29
7	360	89	8	12485.80	Defryn and Sörensen (2017)	12563.80	12725.30	0.1	4
8	440	109	10	13023.20	LMNS	13023.20	13100.70	0.1	38
9	255	52	15	708.18	LMNS	708.18	708.26	0.1	9
10	323	57	17	904.05	LMNS	904.38	904.87	0.3	10
11	399	64	19	1131.09	LMNS	1131.51	1132.21	0.6	17
12	483	70	21	1374.56	LMNS	1374.76	1375.94	1.3	24
13	252	99	26	1015.53	LMNS	1015.53	1024.66	0.1	2
14	320	113	30	1301.35	LMNS	1301.35	1311.66	0.1	5
15	396	124	34	1655.25	LMNS	1655.94	1666.50	0.1	6
16	480	138	38	2038.76	LMNS	2038.76	2044.60	0.1	9
17	240	134	22	817.81	LMNS	817.81	821.42	0.1	18
18	300	175	27	1072.42	LMNS	1072.42	1080.84	0.1	24
19	360	211	33	1505.53	LMNS	1505.53	1511.53	0.1	59
20	420	253	38	2004.59	LMNS	2005.96	2012.76	0.1	80

Table 2.15: Detailed results for `Golden-Exp` instances with $\beta = 0.1$.

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
1	240	41	10	6041.96	LMNS	6041.96	6042.60	0.1	8
2	320	41	10	9672.53	LMNS	9672.53	9674.82	0.1	11
3	400	39	10	13562.80	LMNS	13562.80	13570.00	0.9	19
4	480	42	10	16796.60	LMNS	16796.60	16801.30	2.3	23
5	200	20	5	9328.45	LMNS	9328.45	9328.45	0.9	6
6	280	28	7	10818.30	LMNS	10818.30	10818.30	0.8	9
7	360	35	9	12243.70	LMNS	12243.70	12243.70	0.9	17
8	440	43	11	13998.70	LMNS	13998.70	14007.30	1.2	24
9	255	52	15	712.49	LMNS	713.03	713.74	0.1	7
10	323	57	17	903.94	LMNS	903.94	904.50	0.3	12
11	399	64	19	1125.91	LMNS	1126.38	1126.88	0.7	18
12	483	70	21	1377.69	LMNS	1377.69	1380.84	1.1	24
13	252	99	26	1024.75	LMNS	1024.75	1030.16	0.1	2
14	320	113	30	1290.05	LMNS	1290.05	1297.63	0.1	7
15	396	125	34	1639.93	LMNS	1646.75	1654.94	0.1	8
16	480	138	38	2031.86	LMNS	2031.86	2044.90	0.1	6
17	240	99	22	782.99	LMNS	782.99	783.76	0.1	16
18	300	129	27	1107.43	LMNS	1114.47	1121.61	0.1	4
19	360	154	33	1511.74	LMNS	1511.74	1516.24	0.1	29
20	420	173	38	2015.65	LMNS	2015.65	2023.60	0.1	17

Table 2.16: Detailed results for `Golden-Exp` instances with $\beta = 0.25$.

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
1	240	20	10	6530.59	LMNS	6530.59	6530.59	1.3	4
2	320	20	10	9754.86	LMNS	9754.86	9754.86	3.6	8
3	400	20	9	13121.40	LMNS	13121.40	13121.40	10.3	15
4	480	21	10	17375.50	LMNS	17375.50	17375.50	16.7	25
5	200	11	5	8597.31	Defryn and Sörensen (2017)	8597.30	8597.30	5.3	7
6	280	14	7	10502.70	LMNS	10502.70	10502.70	7.1	11
7	360	18	9	12582.60	LMNS	12582.60	12582.60	10.0	16
8	440	21	10	13635.50	LMNS	13635.50	13635.50	12.7	21
9	255	30	15	695.26	LMNS	695.26	695.26	1.2	6
10	323	34	16	890.87	Defryn and Sörensen (2017)	907.83	907.83	2.0	7
11	399	38	18	1101.44	LMNS	1101.44	1101.47	3.7	14
12	483	41	20	1310.81	LMNS	1310.81	1310.81	5.8	23
13	252	59	27	1048.40	LMNS	1048.40	1048.53	0.1	5
14	320	67	31	1329.28	LMNS	1329.28	1329.42	0.1	13
15	396	74	35	1641.94	LMNS	1641.94	1643.00	0.2	20
16	480	81	39	1986.68	LMNS	1986.68	1988.75	0.4	33
17	240	48	24	880.36	LMNS	880.36	880.36	0.1	9
18	300	60	29	1192.67	LMNS	1192.67	1192.67	0.1	12
19	360	70	34	1612.33	Defryn and Sörensen (2017)	1612.64	1612.64	0.1	14
20	420	82	41	2269.14	LMNS	2269.14	2269.47	0.1	40

Table 2.17: Detailed results for `Golden-Exp` instances with $\beta = 0.5$.

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
1	240	13	12	6736.15	Expósito-Izquierdo *et al.* (2016)	6736.17	6736.17	5.4	6
2	320	14	13	10204.30	Expósito-Izquierdo *et al.* (2016)	10204.30	10204.30	14.1	15
3	400	14	12	13575.70	Expósito-Izquierdo *et al.* (2016)	13577.60	13577.60	40.9	43
4	480	14	13	17077.59	Expósito-Izquierdo *et al.* (2016)	17078.80	17078.80	61.4	64
5	200	8	6	8664.94	Expósito-Izquierdo *et al.* (2016)	8664.93	8664.93	19.4	20
6	280	10	9	11452.01	Expósito-Izquierdo *et al.* (2016)	11452.00	11452.00	25.6	26
7	360	12	11	12901.41	Expósito-Izquierdo *et al.* (2016)	12901.40	12901.40	23.2	33
8	440	15	13	13882.20	LMNS	13882.20	13882.20	42.4	45
9	255	21	19	773.39	Expósito-Izquierdo *et al.* (2016)	773.34	773.34	4.3	6
10	323	23	21	1000.51	Expósito-Izquierdo *et al.* (2016)	1000.45	1000.45	7.9	9
11	399	26	24	1223.66	Expósito-Izquierdo *et al.* (2016)	1223.59	1223.59	14.1	17
12	483	28	26	1475.68	Expósito-Izquierdo *et al.* (2016)	1475.64	1475.64	21.1	25
13	252	37	36	1183.12	Expósito-Izquierdo *et al.* (2016)	1183.10	1183.10	0.4	2
14	320	43	41	1520.55	Defryn and Sörensen (2017)	1520.53	1520.53	0.8	3
15	396	48	46	1825.29	Defryn and Sörensen (2017)	1825.16	1825.16	1.5	5
16	480	52	51	2265.54	Expósito-Izquierdo *et al.* (2016)	2265.51	2265.51	2.6	7
17	240	31	30	1001.02	Expósito-Izquierdo *et al.* (2016)	1001.01	1001.01	0.1	1
18	300	39	37	1392.15	Defryn and Sörensen (2017)	1392.14	1392.14	0.1	2
19	360	47	45	1951.77	Defryn and Sörensen (2017)	1951.76	1951.76	0.1	2
20	420	54	52	2540.22	Expósito-Izquierdo *et al.* (2016)	2540.21	2540.21	0.2	3

Table 2.18: Detailed results for `Golden-Exp` instances with $\beta = 0.75$.

Instance	n	N	m	BKS	*First found by*	LMNS			
						Best	Avg.	T_p	T
1	240	10	9	6293.04	Expósito-Izquierdo *et al.* (2016)	6293.06	6293.06	10.7	11
2	320	11	10	9879.59	Expósito-Izquierdo *et al.* (2016)	9879.60	9879.60	29.7	31
3	400	10	9	12361.09	Expósito-Izquierdo *et al.* (2016)	12366.40	12366.40	82.4	84
4	480	11	10	16130.39	Expósito-Izquierdo *et al.* (2016)	16148.50	16148.50	135.0	137
5	200	6	5	8394.11	Expósito-Izquierdo *et al.* (2016)	8399.08	8399.08	42.9	43
6	280	8	7	10777.33	Expósito-Izquierdo *et al.* (2016)	10777.30	10777.30	61.6	62
7	360	9	8	11346.11	Expósito-Izquierdo *et al.* (2016)	11365.30	11365.30	88.6	90
8	440	11	10	13188.94	Expósito-Izquierdo *et al.* (2016)	13197.30	13197.30	99.4	101
9	255	16	14	705.19	Expósito-Izquierdo *et al.* (2016)	705.15	705.15	9.5	10
10	323	17	16	837.52	Expósito-Izquierdo *et al.* (2016)	837.46	837.46	17.2	18
11	399	19	18	1054.13	Expósito-Izquierdo *et al.* (2016)	1054.74	1054.74	27.6	30
12	483	21	20	1297.31	Expósito-Izquierdo *et al.* (2016)	1297.61	1297.61	49.3	41
13	252	28	27	996.36	Expósito-Izquierdo *et al.* (2016)	996.35	996.35	1.1	2
14	320	31	30	1223.09	Expósito-Izquierdo *et al.* (2016)	1223.07	1223.07	2.2	4
15	396	36	34	1531.29	Expósito-Izquierdo *et al.* (2016)	1531.27	1531.27	3.9	6
16	480	39	38	1874.69	Expósito-Izquierdo *et al.* (2016)	1874.67	1874.67	5.6	9
17	240	23	22	844.27	Expósito-Izquierdo *et al.* (2016)	844.26	844.26	0.9	1
18	300	29	28	1212.97	Expósito-Izquierdo *et al.* (2016)	1212.96	1212.96	1.1	2
19	360	35	34	1667.45	Expósito-Izquierdo *et al.* (2016)	1667.45	1667.45	1.2	3
20	420	40	39	2128.6	Expósito-Izquierdo *et al.* (2016)	2128.58	2128.58	1.3	4

Table 2.19: Detailed results for `Golden-Exp` instances with $\beta = 1.0$.

2.C Details on Significance Tests used in Sections 2.4.1 and 2.4.3

Tables 2.20–2.25 report APVs calculated by Finner's and Shaffer's post-hoc procedures for the significance tests used in Sections 2.4.1 and 2.4.3, respectively. In general, these tests can be applied to detect the specific differences between two algorithms/settings to be significant, if the Friedman's test rejects the null hypothesis stating that all medians between all regarded algorithms/settings are equal. In each table, the considered algorithms/settings are sorted by their ranks in the Friedman test starting with the algorithm/setting ranked best. Note that, for Finner's procedure, the best setting is used as control method, and APVs are given for comparison between the control method and every other setting. Shaffer's procedure does a pairwise comparison among all methods and the corresponding tables report APVs for every pair of algorithms/settings.

Tables 2.20 and 2.21 have the parameter setting *All Operators* as the control method and compare different settings w.r.t. *Gap Best*. The benchmarks taken into account are `GVRP-GC`, `Golden-Bat`, and `Li`.

Tables 2.22 and 2.23 show the APVs calculated by Shaffer's procedure for *Gap Best* and the computation time T, respectively, using the instance set `Golden-Exp`. The algorithms compared are $\text{LMNS}^{3,3}_{5\,000}$ and $\text{LMNS}^{5,0}_{5\,000}$ as well as the methods by Expósito-Izquierdo *et al.* (2016) ('Expósito') and Defryn and Sörensen (2017) ('Defryn').

Similarly, Tables 2.24 and 2.25 show the APVs for the `Golden-Bat` instances.

Settings sorted by rank	APVs of Finner's procedure
Related-Best	0.137
Related-Nearest	0.116
Random-Nearest	0.103
Random-Best	0.005
Worst-Nearest	0.004
Worst-Best	<0.001
Route-Nearest	<0.001
Route-Best	<0.001

Table 2.20: APVs of Finner's procedure for *Gap Best* using the setting *All Operators* as control method and comparing to settings using only one destroy and one repair operator (denoted by *destroy-repair*).

Settings sorted by rank	APVs of Finner's procedure
w/o Random	0.586
w/o Related	0.333
w/o Route	0.248
w/o Worst	0.221
w/o Nearest	<0.001
w/o Best	<0.001

Table 2.21: APVs of Finner's procedure for *Gap Best* using the setting *All Operators* as the control method and comparing to settings without the respective destroy or repair operator. Note that instances where each and every setting finds the BKS are not considered here.

Algorithms sorted by rank	$\text{LMNS}^{3,3}_{5\,000}$	$\text{LMNS}^{5,0}_{5\,000}$	Defryn	Expósito
$\text{LMNS}^{3,3}_{5\,000}$	-	1.244	<0.001	<0.001
$\text{LMNS}^{5,0}_{5\,000}$	-	-	<0.001	<0.001
Defryn	-	-	-	1.244
Expósito	-	-	-	-

Table 2.22: APVs of Shaffer's procedure for the *Gap Best* using the `Golden-Exp` instances.

Algorithms sorted by rank	$\text{LMNS}^{5,0}_{5\,000}$	$\text{LMNS}^{3,3}_{5\,000}$	Defryn	Expósito
$\text{LMNS}^{5,0}_{5\,000}$	-	<0.001	<0.001	<0.001
$\text{LMNS}^{3,3}_{5\,000}$	-	-	<0.001	<0.001
Defryn	-	-	-	1.0
Expósito	-	-	-	-

Table 2.23: APVs of Shaffer's procedure for the average runtime T using the `Golden-Exp` instances.

Algorithms sorted by rank	$\text{LMNS}^{5,0}_{50}$	$\text{LMNS}^{3,3}_{50}$	Defryn
$\text{LMNS}^{5,0}_{50}$	-	0.253	<0.001
$\text{LMNS}^{3,3}_{50}$	-	-	<0.001
Defryn	-	-	-

Table 2.24: APVs of Shaffer's procedure for the *Gap Best* using the `Golden-Bat` instances.

Algorithms sorted by rank	$\text{LMNS}^{5,0}_{50}$	$\text{LMNS}^{3,3}_{50}$	Defryn
$\text{LMNS}^{5,0}_{50}$	-	0.116	0.002
$\text{LMNS}^{3,3}_{50}$	-	-	0.063
Defryn	-	-	-

Table 2.25: APVs of Shaffer's procedure for the average runtime T using the `Golden-Bat` instances.

Chapter 3

Exact Solution of the Soft-Clustered Vehicle-Routing Problem

Timo Hintsch, Stefan Irnich

Abstract

The soft-clustered vehicle-routing problem (SoftCluVRP) extends the classical capacitated vehicle-routing problem by one additional constraint: The customers are partitioned into clusters and feasible routes must respect the soft-cluster constraint, that is, all customers of the same cluster must be served by the same vehicle. In this chapter, we design and analyze different branch-and-price algorithms for the exact solution of the SoftCluVRP. The algorithms differ in the way the column-generation subproblem, a variant of the shortest-path problem with resource constraints (SPPRC), is solved. The standard approach for SPPRCs is based on dynamic-programming labeling algorithms. We show that even with all the recent acceleration techniques (e.g., partial pricing, bidirectional labeling, decremental state space relaxation) available for SPPRC labeling algorithms, the solution of the subproblem remains extremely difficult. The main contribution is the modeling and solution of the subproblem using a branch-and-cut algorithm. The conducted computational experiments prove that branch-and-price equipped with this integer programming-based approach outperforms sophisticated labeling-based algorithms by one order of magnitude. The largest SoftCluVRP instances solved to optimality have more than 400 customers or more than 50 clusters.

3.1 Introduction

The *clustered vehicle-routing problem* (CluVRP, Sevaux and Sörensen, 2008) is a variant of the classical *capacitated vehicle-routing problem* (CVRP, Toth and Vigo, 2014) in which the customers are grouped into disjoint clusters. A feasible CluVRP route must serve each cluster integrally, that is, all customers of a cluster must be served by the same vehicle and in consecutive visits. This problem has been approached by exact optimization algorithms (Battarra *et al.*, 2014a) as well as metaheuristics (Barthélemy *et al.*, 2010; Expósito Izquierdo *et al.*, 2013; Vidal *et al.*, 2015; Expósito-Izquierdo *et al.*, 2016; Defryn and Sörensen, 2017; Pop *et al.*, 2018; this thesis, Chapter 2). We consider a relaxation of the CluVRP, the *soft-clustered vehicle-routing problem* (SoftCluVRP), in which the only additional constraint compared to the CVRP is that all customers of a cluster must be served by the same vehicle. In contrast to the CluVRP, visits to customers of the same cluster may or may not be interrupted by visits to other customers. The SoftCluVRP has been introduced by Defryn and Sörensen (2017).

CluVRP and SoftCluVRP arise in applications where the routing decision must take into account already taken clustering decisions. For example, Sevaux and Sörensen (2008) mention parcel/small-package delivery in courier companies as an application field: In a first step, parcels are sorted according to given districting (see Butsch *et al.*, 2014, for districting). Typically, the sorting policy, i.e., the sorting of parcels e.g. by regional zones and/or ZIP codes, is altered only once in a while. Note that the sorting policy must always be fixed before the actual demand distribution over the zones is known. In a second step, the batches of parcels, as they result from the sorting, are delivered to the recipients.

An instance of the SoftCluVRP is defined over a complete undirected graph $G = (V, E)$ with the vertex set $V = \{0, \dots, n\}$ and the edge set E. The vertex set comprises the depot vertex 0 and the customer vertices $V \setminus \{0\} = \{1, \dots, n\}$. The vertices are partitioned into $N + 1$ clusters $V_0, V_1, V_2, \dots, V_N$, where we define the depot cluster $V_0 = \{0\}$ for convenience. The *customer clusters* are indexed by $h \in H = \{1, 2, \dots, N\}$ and a positive demand $d_h > 0$ is associated with cluster V_h, while the depot cluster V_0 has zero demand $d_0 = 0$. For $h \in H \cup \{0\}$, the cardinality of a cluster is $n_h = |V_h|$ and $h(i) \in H \cup \{0\}$ refers to the index of the cluster to which vertex $i \in V$ belongs. A homogeneous fleet of m vehicles with capacity Q is hosted at the depot 0. Non-negative routing costs c_{ij} are associated with every edge $\{i, j\} \in E$.

A *route* $r = (i_0, i_1, \dots, i_r, i_{r+1})$ is a cycle in G passing through the depot 0, i.e., a cycle with $i_0 = i_{r+1} = 0$. The customer clusters *touched* by the route r, i.e., with $V_h \cap \{i_1, \dots, i_r\} \neq \varnothing$, are denoted by $H(r) \subseteq H$. The route r is feasible for the SoftCluVRP if

(i) $i_1, \dots, i_r$ are all different, (elementarity constraint)

(ii) for all $h \in H(r)$, $V_h \subseteq \{i_1, \ldots, i_r\}$, (soft-cluster constraints)
(iii) and $\sum_{h \in H(r)} d_h \leq Q$. (capacity constraint)

Note that for the CluVRP the constraints (ii) must be replaced by the following conditions: for all $h \in H(r)$, there exists an index $k \in \{1, 2, \ldots, r - n_h + 1\}$ such that $V_h = \{i_k, i_{k+1}, \ldots, i_{k+n_h-1}\}$ (hard-cluster constraints).

The contribution of this chapter is the design and computational analysis of different branch-and-price algorithms for the exact solution of the SoftCluVRP. Branch-and-price is the leading solution approach for many variants of the VRP (recently surveyed by Irnich *et al.*, 2014) and can be summarized as follows: A route/path-based extended formulation of the VRP variant, the so-called *master program*, is solved via *column generation* (Desaulniers *et al.*, 2005). The starting point is always a *restricted master program* (RMP) that comprises a (small) subset of routes and relaxes the integrality constraints on the route variables (integrality is later enforced via branching). Missing routes for the solution of the linear relaxation of the master program are dynamically and iteratively generated with the help of a pricing subproblem. This pricing problem can be formulated as a *shortest-path problem with resource constraints* (SPPRC, Irnich and Desaulniers, 2005). For almost all VRP variants, the associated SPPRCs are best solved with dynamic-programming labeling algorithms (e.g. Feillet *et al.*, 2004; Irnich and Villeneuve, 2006; Righini and Salani, 2008; Baldacci *et al.*, 2011). To our surprise, the SPPRCs that result from the SoftCluVRP are extremely difficult to solve even for instances of rather moderate size. This chapter will show that even with all the recent acceleration techniques available for the SPPRC dynamic-programming labeling algorithms, the situation hardly improves. For example, bidirectional labeling has proven very effective for many VRP variants (Righini and Salani, 2006; Tilk *et al.*, 2017) but the soft-cluster constraints are so loose constraints that the combinatorial explosion is often not effectively suppressed. Gschwind *et al.* (2017) report a similar phenomenon for some loosely-constrained VRPs with pickup-and-delivery structure. Our most important finding is, therefore, that a relatively simple *integer programming* (IP) formulation solved with a standard IP-solver is highly effective for the SoftCluVRP pricing problems. This result is rather remarkable because IP-based approaches such as branch-and-cut have almost never reached the performance of labeling-based pricing algorithms (we refer to the discussion in Drexl and Irnich, 2012).

The remainder of this chapter is structured as follows. Section 3.2 provides a compact three-index formulation for the SoftCluVRP that we use to derive the path-based reformulation and the pricing subproblem. Section 3.3 presents the two exact solution approaches for the pricing subproblem that are based on dynamic-programming labeling and branch-and-cut. Moreover, a primal heuristic pricing algorithm tailored to the SoftCluVRP is presented. The overall pricing strategies

as well as branching and its impact on the subproblem is discussed in Section 3.4. Section 3.5 summarizes the comprehensive computational studies conducted on the new branch-and-price algorithms. Final conclusions are drawn in Section 3.6.

3.2 Three-Index, Extensive, and Subproblem Formulation

A three-index formulation for the asymmetric version of the SoftCluVRP was presented by Defryn and Sörensen (2017). In Section 3.2.1, we present another three-index formulation for the symmetric version of the SoftCluVRP, because the available benchmark instances are all symmetric. From this three-index model, we derive an extensive route-based formulation via IP Dantzig-Wolfe decomposition (Lübbecke and Desrosiers, 2005) in Section 3.2.2. Moreover, the associated SPPRC subproblem is formulated as an IP in Section 3.2.3.

For the models and any subset $S \subseteq V$, we use the notation $\delta(S)$ for the set of edges $e = \{i, j\} \in E$ with exactly one endpoint in S, i.e., either $i \in S$ and $j \in V \setminus S$ or $i \in V \setminus S$ and $j \in S$. For singleton sets $S = \{i\}$ with $i \in V$, we write $\delta(i)$ instead of $\delta(\{i\})$. Finally, we define $r(S)$ as the minimum number of vehicles needed to serve the customers $S \subseteq V \setminus \{0\}$. For the models presented subsequently, the lower bound $\lceil (\sum_{h \in H: V_h \cap S \neq \varnothing} d_h)/Q \rceil$ for $r(S)$ is sufficient. As for other purely capacitated VRP variants, exact values $r(S)$ can be determined by solving the corresponding bin-packing problem.

3.2.1 Three-Index Formulation

The following three-index formulation explicitly models the routes performed by each of the m vehicles. We therefore define the *fleet* as $K = \{1, 2, \ldots, m\}$. The three-index formulation uses two types of integer decision variables both indexed by $k \in K$. First, for each edge $e \in E$ and vehicle $k \in K$ there is a routing variable x_e^k indicating how many times vehicle k traverses edge e (leading to three indices i, j, k for $e = \{i, j\} \in E$ and $k \in K$). Note that edges adjacent to the depot 0 may be traversed twice. The cluster-assignment variable z_h^k, one for each cluster index $h \in H$ and vehicle $k \in K$, indicates that vehicle k serves cluster V_h. The model is:

$$\min \sum_{k\in K}\sum_{e\in E} c_e x_e^k \tag{3.1a}$$

$$\text{subject to} \sum_{k\in K} z_h^k = 1 \qquad \forall h \in H \tag{3.1b}$$

$$\sum_{e\in\delta(0)} x_e^k = 2 \qquad \forall k \in K \tag{3.1c}$$

$$\sum_{e\in\delta(i)} x_e^k = 2z_{h(i)}^k \qquad \forall i \in V\setminus\{0\}, \forall k \in K \tag{3.1d}$$

$$\sum_{k\in K}\sum_{e\in\delta(S)} x_e^k \geq 2r(S) \qquad \forall S \subseteq V\setminus\{0\}, S \neq \varnothing \tag{SEC}$$

$$x_e^k \in \{0,1,2\} \qquad \forall e \in \delta(0), \forall k \in K \tag{3.1e}$$

$$x_e^k \in \{0,1\} \qquad \forall e \in E\setminus\delta(0), \forall k \in K \tag{3.1f}$$

$$z_h^k \in \{0,1\} \qquad \forall h \in H, \forall k \in K \tag{3.1g}$$

The objective (3.1a) minimizes the routing costs over all vehicles. Constraints (3.1b) ensure that each cluster is served exactly once. The fleet size is set to m by condition (3.1c). Replacing $=$ by $\leq$ in (3.1c) allows modeling that less than m vehicles can be used. Note that for some benchmark problems considered in Section 3.5 the parameter m is chosen larger than the minimum fleet size (to be computed by solving a bin packing problem with the cluster demands and bins of size Q), but that empty routes are not allowed in these instances. The routing and cluster-assignment variables are coupled via (3.1d) making sure that each customer $i \in V_h$ (for each $h \in H$) is served by vehicle k, i.e., $\sum_{e\in\delta(i)} x_e^k = 2$, if and only if the cluster V_h is assigned to that vehicle k. The exponentially-sized family of *subtour-elimination constraints* is given by (SEC). Note that these *aggregated* subtour-elimination constraints (SEC) can also be written in *disaggregated* form as

$$\sum_{e\in\delta(S)} x_e^k \geq 2z_{h(i)}^k \qquad \forall S \subseteq V\setminus\{0\}, S \neq \varnothing, \forall i \in S, \forall k \in K \tag{3.1h}$$

together with explicit *capacity constraints*

$$\sum_{h\in H} d_h z_h^k \leq Q \qquad \forall k \in K. \tag{3.1i}$$

With this replacement, i.e., (SEC) replaced by (3.1h) and (3.1i), the only constraints of the model (in the following referred to as model (3.1)) that are not vehicle-specific are the constraints (3.1b).

3.2.2 Extensive Route-Based Formulation

An IP Dantzig-Wolfe decomposition on the non-vehicle specific constraints (3.1b) and a subsequent aggregation over the vehicles $k \in K$ leads to an *extensive formulation* with one binary decision variable for each feasible route (see Lübbecke and Desrosiers, 2005). Let Ω be the set of all feasible SoftCluVRP routes fulfilling conditions (i)–(iii) of Section 3.1 (feasible w.r.t. elementarity, soft-cluster, and capacity constraints). Let y_r be the binary decision variable indicating whether route $r \in \Omega$ is chosen in the solution or not. Moreover, let c_r be the cost of route r defined as the sum of the routing costs for all edges traversed by r. Finally, we define a_{hr} as the binary indicator whether route r serves cluster V_h or not.

The extensive path-based formulation is the following extended set-partitioning model:

$$\min \sum_{r \in \Omega} c_r y_r \tag{3.2a}$$

$$\text{subject to} \sum_{r \in \Omega} a_{hr} y_r = 1 \qquad \forall h \in H \tag{3.2b}$$

$$\sum_{r \in \Omega} y_r = m \tag{3.2c}$$

$$y_r \in \{0, 1\} \qquad \forall r \in \Omega \tag{3.2d}$$

The objective (3.2a) minimizes the total routing costs. The partitioning constraints (3.2b) ensure that each cluster is served exactly once. Condition (3.2c) is the aggregated convexity constraint of the Dantzig-Wolfe decomposition saying that the number of routes to choose in the solution is m. Also here replacing $=$ by $\leq$ would allow improved solutions in case that m is larger than the minimum fleet size, without changing the suggested branch-and-price solution approaches.

For a subset $\bar{\Omega} \subset \Omega$ of the routes, the linear relaxation of (3.2) defined over $\bar{\Omega}$ is the corresponding RMP. Missing routes are determined by solving the following pricing subproblem.

3.2.3 Subproblem Formulation

The task of the subproblem is to identify negative reduced-cost variables (i.e., routes) or to prove that there exist none. Let π_h for $h \in H$ be the dual prices (a.k.a. shadow prices) of the partitioning constraints (3.2b) and let μ be the dual price of the fleet-size constraint (3.2c). Since we assume the triangle inequality to hold for the costs (c_{ij}), the partitioning constraints (3.2b) can always be replaced by covering constraints, i.e., inequalities with ≥ 1. Therefore, $\pi_h \geq 0$ can be assumed for all $h \in H$.

Recall that for any feasible route $r \in \Omega$, the clusters served by route r are given by the indices $h \in H(r)$. The set $H(r)$ can be written as $\{h \in H : a_{hr} = 1\}$. Hence, the reduced cost of a route r denoted by $\tilde{c}_r$ is given by $c_r - \sum_{h \in H} a_{hr}\pi_h - \mu$.

Note first that the optimal routing cost c_r can be computed by solving a *traveling salesman problem* (TSP, Gutin and Punnen, 2007) over the vertex set $V(r) = V_0 \cup \bigcup_{h \in H(r)} V_h$. Note second that $\sum_{h \in H} a_{hr}\pi_h = \sum_{h \in H(r)} \pi_h$ and that μ is independent of the route r.

The formulation of the subproblem can be formally derived from the original three-index formulation (3.1) by (i) dropping the vehicle index $k \in K$ from the routing variables x and cluster-assignment variables z, (ii) incorporating the dual prices into the objective, and (iii) leaving out the non-vehicle-specific constraints (3.1b). The resulting model has integer routing variables x_e describing how many times edges $e \in E$ are traversed and binary variables z_h for the selection of the served clusters $h \in H$:

$$\tilde{c}(\pi_h, \mu) = \min \sum_{e \in E} c_e x_e - \sum_{h \in H} \pi_h z_h - \mu \tag{3.3a}$$

$$\text{subject to} \sum_{e \in \delta(0)} x_e = 2 \tag{3.3b}$$

$$\sum_{e \in \delta(i)} x_e = 2z_{h(i)} \qquad \forall i \in V \tag{3.3c}$$

$$\sum_{e \in \delta(S)} x_e \geq 2z_{h(i)} \qquad \forall S \subseteq V \setminus \{0\}, S \neq \varnothing, i \in S \tag{3.3d}$$

$$\sum_{h \in H} d_h z_h \leq Q \tag{3.3e}$$

$$x_e \in \{0, 1, 2\} \qquad \forall e \in \delta(0) \tag{3.3f}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \setminus \delta(0) \tag{3.3g}$$

$$z_h \in \{0, 1\} \qquad \forall h \in H \tag{3.3h}$$

The objective (3.3a) is the minimization of the reduced cost. Constraint (3.3b) ensures that the route starts and ends at the depot, and constraints (3.3c) couple the routing decision with the cluster selection, i.e., customer $i \in V \setminus \{0\}$ is visited if and only if its cluster $V_{h(i)}$ is selected. Subtour-elimination constraints are given by (3.3d). Constraint (3.3e) is the capacity constraint. Note that these constraints (3.3b)–(3.3e) are derived from constraints (3.1c), (3.1d), (3.1h), and (3.1i), respectively.

The subproblem (3.3) generalizes the well known *two-matching model* of the TSP (see, e.g., Gutin and Punnen, 2007). Indeed, without the capacity constraint (3.3e) and for large positive dual prices $\pi_h \gg 0$, an optimal solution would select all

clusters (i.e., $z_h = 1$ for all $h \in H$) so that (3.3b) and (3.3c) become the two-matching constraints. With the capacity constraints, the subproblem is a *TSP with profits* (Feillet *et al.*, 2005), even if profits π_h are associated to clusters and not individual customers/vertices. According to the classification by Feillet *et al.* (2005), on the one hand, the subproblem is a *profitable tour problem* because it combines the routing cost minimization and the maximization of the cluster profits π_h in its objective. On the other hand, the capacity constraint imposes a lower bound on the not-chosen clusters. Considering the not-selected clusters introduces a penalty for not choosing clusters into the objective function so that the subproblem can also be characterized as a *prize-collecting TSP* (in the sense of Balas, 1989).

3.3 Solution of the Subproblem

In this section, we describe the two competing solution methods for the solution of the pricing subproblem (3.3). We start with the traditional and established solution approach based on dynamic-programming labeling in Section 3.3.1. Our goal is to concisely define the basic components of a SoftCluVRP-tailored labeling algorithm (definition of attributes, initial label, label extension, and feasibility conditions as well as the dominance principle). Moreover, we refine the basic labeling algorithm by integrating the most recent algorithmic enhancements including bidirectional labeling, several acceleration techniques, and the adaptations for the *ng*-path relaxation. The most important novelty of this chapter is however the presentation of an IP-based solution approach in Section 3.3.2. Our branch-and-cut algorithm is based directly on formulation (3.3) for which we present the initial model and separation procedures for cutting off infeasible integer solutions as well as for finding violated inequalities in fractional solutions. Finally, we developed a direct primal heuristic solver for SoftCluVRP subproblems that searches for negative reduced-cost routes by applying cluster-based add- and drop-exchanges. The primal heuristic solver is used as a heuristic acceleration technique in conjunction with the labeling or branch-and-cut algorithms. It is summarized in Section 3.3.3.

The course of our column-generation algorithm is summarized in Algorithm 3, for which all its components are explained in the remainder of this chapter. In order to lighten the presentation, we omit all termination conditions in Algorithm 3 and assume that the main loop (Steps 2 to 20) over the hierarchy of heuristic and one final exact pricing algorithm is stopped whenever a negative reduced cost route has been found. The pseudocode and references to sections may also serve the reader to better follow the exposition of the column-generation method and to see the independencies between the algorithmic components.

Algorithm 3: `Column Generation`

Input: Dual prices $(\pi_h)_{h\in H}, \mu$, primal solution $\bar{y}_r$, $r \in \bar{\Omega}$
Output: One or several negative reduced cost paths (if any exist)

Build hierarchy $\mathcal{A}$ of heuristic and (one) exact pricing algorithms `// Sections 3.4.1, 3.5.2, and 3.5.3`
for $A \in \mathcal{A}$ **do**
  if *A is primal heuristic* **then**
    for $r \in \bar{\Omega}$ *with* $\bar{y}_r > 0$ **do**
      Call `PrimalHeuristic`(r) `// Section 3.3.3, Algorithm 5`
  if *A is labeling based* **then** `// Section 3.3.1`
    Select reduction level `reduce` $\in \{true, false\}$ `// Section 3.3.1.3`
    Call `Preprocessing(reduce)` `// Algorithm 4`
    Build reduced network (`network size`) `// Section 3.3.1.4`
    Select *dir* $\in$ {`monodirectional`, `bidirectional`} `// Sections 3.3.1.1 and 3.3.1.2`
    Select dominance rule *Rule* $\in$ {`Rule 2`, `Rule 3`}
    if *A uses DSSR* **then** `// Section 3.3.1.4`
      repeat
        Initialize or enlarge *ng*-neighborhoods
        Solve *ng*-path relaxation with labeling $(dir, Rule)$
      until *solution is elementary*
    else
      Solve elementary SPPRC with labeling$(dir, Rule)$ `// Sections 3.3.1.1 and 3.3.1.2`
  if *A is branch-and-cut based* **then** `// Section 3.3.2`
    Solve model (3.3) with branch-and-cut `// Sections 3.2.3 and 3.3.2`

3.3.1 Labeling Algorithms

In this subsection, we define the SPPRC on the corresponding directed graph $G' = (V', A)$. The vertex set V' comprises all customer vertices $V \setminus \{0\}$ and two copies of the depot denoted by 0 and $0'$, where the origin depot 0 has no ingoing arcs and the destination depot $0'$ has no outgoing arcs. Hence, the arc set is $A = \{(i,j) : i,j \in V \setminus \{0\}, i \neq j\} \cup \{(0,i) : i \in V \setminus \{0\}\} \cup \{(i,0') : i \in V \setminus \{0\}\}$. Recall that a route r in G was defined as a cycle through 0. Any (feasible) route r in G imposes a 0-$0'$-path $r' = (i_0, i_1, \ldots, i_r, i_{r+1})$ in G' (fulfilling conditions (i)–(iii) of Section 3.1), and vice versa.

3.3.1.1 Monodirectional Labeling

A forward monodirectional labeling algorithm starts with an initial label at the origin depot 0 that represents the partial path (0). It propagates labels over arcs toward the destination depot $0'$ with the help of so-called *resource extension functions* (REFs, Desaulniers *et al.*, 1998). Each label stores the resource consumption of the corresponding partial path $(0, ..., i)$ that starts at 0 and ends at some vertex $i \in V'$. To avoid the enumeration of all feasible partial paths, provably redundant labels are eliminated through a dominance criterion.

In the SoftCluVRP, a partial path $p = (0, \ldots, i)$ is represented by a label L that

has the following attributes a.k.a. resources:

L^{cost}: the accumulated reduced cost of the partial path p;
L^{load}: the total demand of clusters touched by p;
$(L^{rem_h})_{h\in H}$: for each $h \in H$, the number of remaining customers $i \in V_h$ that must be visited by a feasible completion of path p; the initial value is set to -1 to indicate that cluster V_h has not yet been visited;
$(L^{visit_v})_{v\in V\setminus\{0\}}$: the binary visit attributes for customers indicating whether or not customer $v \in V \setminus \{0\}$ has been visited along path p.

The attribute $L_i^{visit_v}$ can also be set to 1 and the attribute $L_i^{rem_h}$ set to 0, if cluster $V_h = V_{h(v)}$ has not yet been visited but it is *unreachable* for all completions of p. The latter condition is fulfilled if the demand d_h exceeds the residual capacity $Q - L_i^{load}$ of p (for details see Feillet *et al.*, 2004). Note also that as long as partial paths are elementary, the attributes $(L^{rem_h})_{h\in H}$ can be computed from the attributes $(L^{visit_v})_{v\in V\setminus\{0\}}$. However, when using the *ng*-path relaxation (see Section 3.3.1.4) this is no longer true. We also keep the attributes $(L^{rem_h})_{h\in H}$, because they simplify the presentation of and actual computations within the labeling algorithms.

The initial label for $p = (0)$ is defined as $L_0 = (L_0^{cost}, L_0^{load}, L_0^{rem}, L_0^{visit}) = (0, 0, (-\mathbf{1}), (\mathbf{0}))$, where $-\mathbf{1}$ and $\mathbf{0}$ are vectors with all entries equal to -1 and 0, respectively, of appropriate size.

Next we describe the extension of a label L_i belonging to a feasible partial path $(0, \ldots, i)$ along an arc $(i, j) \in A$ to vertex j. The REFs create a new label $L_j = (L_j^{cost}, L_j^{load}, L_j^{rem}, L_j^{visit})$ with the following attributes:

$$L_j^{cost} = L_i^{cost} + c_{ij} - \begin{cases} \pi_{h(j)}, & \text{if } L_i^{rem_{h(j)}} = -1 \\ 0, & \text{otherwise} \end{cases} - \begin{cases} \mu/2, & \text{if } i = 0 \text{ or } j = 0' \\ 0, & \text{otherwise} \end{cases} \tag{3.4a}$$

$$L_j^{load} = L_i^{load} + \begin{cases} d_{h(j)}, & \text{if } L_i^{rem_{h(j)}} = -1 \\ 0, & \text{otherwise} \end{cases} \tag{3.4b}$$

$$L_j^{rem_h} = \begin{cases} n_h - 1, & \text{if } h = h(j) \text{ and } L_i^{rem_{h(j)}} = -1 \\ L_i^{rem_h} - 1, & \text{if } h = h(j) \text{ and } L_i^{rem_{h(j)}} > 0 \\ L_i^{rem_h}, & \text{otherwise} \end{cases} \qquad \forall h \in H \tag{3.4c}$$

$$L_j^{visit_v} = \begin{cases} L_i^{visit_v} + 1, & \text{if } v = j \\ L_i^{visit_v}, & \text{otherwise} \end{cases} \qquad \forall v \in V \setminus \{0\} \tag{3.4d}$$

The condition $L_i^{rem_{h(j)}} = -1$ in (3.4a), (3.4b), and (3.4c) tests whether the next visit to vertex j is one to a non-visited cluster $V_{h(j)}$. Accordingly, the dual prices $\pi_{h(j)}$ and demands $d_{h(j)}$ are incorporated only if $V_{h(j)}$ has not been visited yet.

The new label L_j for $j \in V \setminus \{0, 0'\}$ and the associated partial path $(p, j) = (0, \ldots, i, j)$ is feasible if and only if

$$L_j^{load} \leq Q \qquad \text{and} \qquad L_j^{visit_j} \leq 1, \forall j \in V \setminus \{0\}. \tag{3.5a}$$

For an extension to the destination depot $j = 0'$, the additional feasibility conditions

$$L_j^{rem_h} \leq 0 \qquad \forall h \in H \tag{3.5b}$$

are needed to guarantee that no cluster is served incompletely.

Weak Dominance Let L and L' be two labels of different partial paths p and p', respectively, that end at the same vertex i. Domination of labels generally uses the following *auxiliary criterion*: If for each feasible completion q' of p' into a feasible 0-0′-path $r' = (p', q')$ there exists a feasible completion q of p into a feasible 0-0′-path $r = (p, q)$ and the reduced cost of r is not greater than the reduced cost of r', then label L' is *dominated* by L. A dominated label can be discarded (if the dominating label is kept).

For many VRP variants, a simplified criterion is applied by choosing the completion q identical to the completion q'. Using this simplified criterion is unnecessarily restrictive for the SoftCluVRP as it is also too restrictive in VRPs with pickup-and-delivery structure (see discussion in Section 3.3.1.5). However, the following weak dominance rule can be directly derived from the simplified criterion:

Rule 1. *(Weak Dominance Rule) Let L and L' be two labels of different partial paths that end at the same vertex i. Label $L = (L^{cost}, L^{load}, L^{rem}, L^{visit})$ dominates label $L' = (L'^{cost}, L'^{load}, L'^{rem}, L'^{visit})$ if all of the following conditions are fulfilled:*

$$L^{cost} \leq L'^{cost} \tag{3.6a}$$

$$L^{load} \leq L'^{load} \tag{3.6b}$$

$$\left\{ \begin{array}{lll} (L^{rem_h} = L'^{rem_h} & and & L_i^{visit_v} = L_i'^{visit_v}, \forall v \in V_h) \\ or\ (L^{rem_h} = -1 & and & L'^{rem_h} = 0) \end{array} \right\} \quad \forall h \in H \tag{3.6c}$$

The conditions (3.6c) ensure that L_i dominates L_i' only if either (i) both partial paths p and p' (p is the path associated with L, p' with L') have visited exactly the same vertices $v \in V_h$ of a cluster or (ii) path p has not visited a cluster V_h that is already completely served by p'.

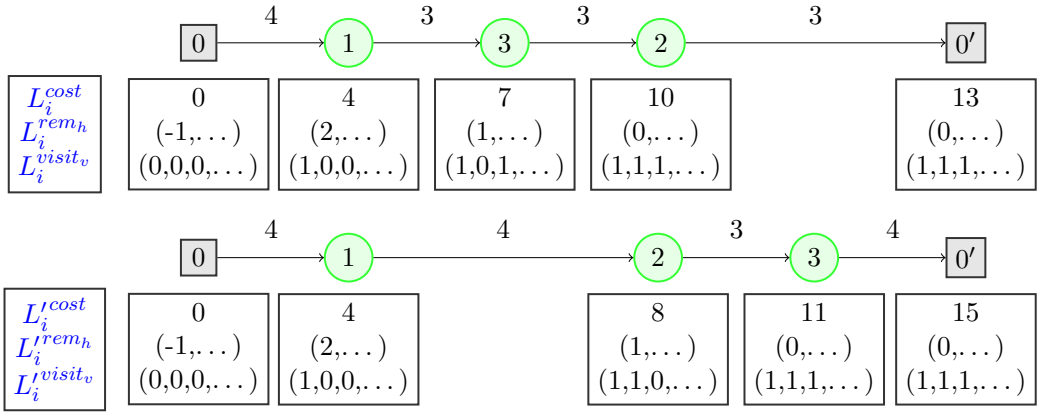

Figure 3.1: Propagation of the attributes for two routes $r = (0, 1, 3, 2, 0')$ and $r' = (0, 1, 2, 3, 0')$ serving the same cluster $V_1 = \{1, 2, 3\}$

Example 1. *We consider a SoftCluVRP instance and two routes that visit only one cluster* $V_1 = \{1, 2, 3\}$*, but customers are visited in different sequences. The relevant part of the (symmetric) cost matrix* (c_{ij}) *is:*

c_{ij}	$0'$	1	2	3
0	–	4	3	4
1	4	–	4	3
2	3	4	–	3
3	4	3	3	–

Let the two routes be $r = (0, 1, 3, 2, 0')$ *and* $r' = (0, 1, 2, 3, 0')$*. Moreover, we assume that the dual prices* π_1 *and* μ *are zero because they are irrelevant for the exposition. Figure 3.1 depicts the propagation of the attributes* L^{cost}, L^{rem_h}*, and* L^{visit_v} *along the routes. The attribute* L^{load} *is only altered along arc* $(0, 1)$ *and not presented in order to keep the example as small as possible.*

Note first that both routes share the same partial path $(0, 1)$*, for which only one label is created, so that there is no dominance between subpaths of* r *and* r' *at vertex* 1*. In contrast, there are two labels at vertex* 2*, say* L_2 *and* L_2' *but with the weak dominance rule neither label dominates the other one. The same holds at vertex* 3 *for the corresponding labels. However, at the destination vertex* $0'$*, the label of* r *dominates the label of* r' *due to its better cost but otherwise identical attributes.*

Strong Dominance A stronger dominance can be achieved if the distance matrix (c_{ij}) respects the *triangle inequality* (TI), i.e., for any three different vertices $i, j, k \in V$ the inequality $c_{ik} \leq c_{ij} + c_{jk}$ holds. We analyze this property in detail because later presented acceleration techniques (see Section 3.3.1.3) modify

the distance matrix so that the TI may become violated. Moreover, widely used benchmark instances for the SoftCluVRP (see Section 3.5.1) round down Euclidean distances so that the TI can be violated.

We assume now that the TI holds. Then, no path completion can benefit from visiting an additional customer. More precisely, for each $h \in H$, let

$$R_h = \{i \in V_h : L^{rem_h} > 0, L^{visit_i} = 0\} \tag{3.7}$$

be the vertices in V_h that *must* be visited by any completion of path p corresponding to L (due to conditions (3.5b)). Similarly, we define $R'_h = \{i \in V_h : L'^{rem_h} > 0, L'^{visit_i} = 0\}$ as the vertices that *must* be visited by any completion of path p' corresponding to L'. If $R_h \subseteq R'_h$, then every feasible extension of p' must visit additional customers compared to a corresponding feasible extension of p. Therefore, the strong dominance rule is:

Rule 2. *(Strong Dominance Rule) Let L and L' be two labels of different partial paths that end at the same vertex i. Label $L = (L^{cost}, L^{load}, L^{rem}, L^{visit})$ dominates label $L' = (L'^{cost}, L'^{load}, L'^{rem}, L'^{visit})$ if all of the following conditions are fulfilled:*

$$\text{(3.6a) } \textit{and } \text{(3.6b)} \tag{3.8a}$$

$$\left\{ \begin{array}{lll} (L^{rem_h} \leq L'^{rem_h} & \textit{and} & L^{visit_v} \geq L'^{visit_v}, \forall v \in V_h) \\ \textit{or} & & L^{rem_h} = -1 \end{array} \right\} \quad \forall h \in H \tag{3.8b}$$

Proof. The proof relies on the above mentioned auxiliary criterion. The partial path associated with L (L') is denoted by p (p'). Let q' be an arbitrary feasible extension of p' into a feasible route $r' = (p', q')$. We will show that there exists a feasible extension q of p so that the route $r = (p, q)$ is feasible and fulfills $\tilde{c}_r \leq \tilde{c}_{r'}$.

Let q be the path that results from the removal of all vertices $\{v \in V : L^{visit_v} = 1, L'^{visit_v} = 0\} \cup \{v \in V : L^{rem_{h(v)}} = -1, L'^{rem_{h(v)}} > 0, L'^{visit_v} = 0\}$ from q'. The REFs (3.4) and feasibility conditions (3.5) guarantee that (p, q) is feasible. Indeed, every cluster that is only partly served with p is at the end served completely with (p, q).

Moreover, the TI ensures that the routing cost of q is not greater than the routing cost of q'. Finally, all clusters completely served with q' are also completely served with q. Consequently, both extensions q and q' collect exactly the same dual prices $\pi_{h(j)}$ in (3.4a). Therefore, the reduced cost of $r = (p, q)$ is not greater than the reduced cost of $r' = (p', q')$, i.e., $\tilde{c}_r \leq \tilde{c}_{r'}$. □

Weak dominance (3.6) allows domination only for labels with $R_h = R'_h$ for all $h \in H$. In contrast, strong dominance (3.8) allows domination also if $R_h \subseteq R'_h$ holds.

Example 2. (cont'd from Example 1) *The stronger dominance Rule 2 is not imposing additional dominance relations in the situation of Example 1. Indeed, while L_2 and L'_2 fulfill conditions* (3.8b) *and* (3.6b)*, they fail on the reduced cost condition* (3.6a)*.*

However, the situation can change with another cost matrix. Assume that instead of (c_{ij}) the cost matrix were $(\dot{c}_{ij})$ defined by:

$\dot{c}_{ij}$	$0'$	1	2	3
0	–	3	2	3
1	3	–	2	1
2	2	2	–	1
3	3	1	1	–

The result of this re-definition is that now both labels L_2 and L'_2 have identical reduced costs $L_2^{cost} = L_2'^{cost} = 5$ so that L_2 dominates L'_2 by Rule 2. We discuss in Section 3.3.1.4 how a systematic modification of the cost matrix can be exploited as an acceleration technique.

3.3.1.2 Bidirectional Labeling

Righini and Salani (2006) coined *bounded bidirectional labeling* for SPPRCs. Numerous subsequent works have shown that bidirectional labeling algorithms are usually superior to their monodirectional counterparts. Accordingly, bidirectional labeling has become a quasi-standard for solving SPPRCs. In addition, two recent works (Tilk *et al.*, 2017; Gschwind *et al.*, 2017) have shown that bidirectional labeling can significantly benefit from a dynamically chosen half-way point which exploits the inherent asymmetry on many SPPRC instances. We briefly explain the ideas of bidirectional labeling in the following.

Bidirectional labeling requires the definition of a monotone resource, i.e., an attribute L^{res} to be used for bounding the propagation of labels in both directions. More precisely, for labels L_{fw} propagated in forward direction, the respective attribute L_{fw}^{res} is increasing, and for labels L_{bw} propagated in backward direction, the attribute L_{bw}^{res} is decreasing along the path. Moreover, for any feasible 0-$0'$-path (p, q) with $p = (0, \ldots, i)$ and associated forward label L_{fw} and $q = (i, \ldots, 0')$ with associated backward label L_{bw}, the attributes must fulfill $L_{fw}^{res} \leq L_{bw}^{res}$. The bounding component of the bidirectional labeling algorithm uses a so-called *half-way point* HWP und only propagates forward labels with $L_{fw}^{res} \leq HWP$ and backward labels with $L_{bw}^{res} > HWP$. Examples of such monotone forward and backward resources are earliest and latest service times for VRPs with time windows, or the accumulated demand and the residual capacity for capacitated VRPs.

In order not to generate multiple copies of the same route (a non-trivial route r has multiple representations $r = (p, q) = (p', q')$ where p is a subpath of p' and q' a subpath of q, or vice versa), the *merge criterion* also relies on the half-way point: For the merge of two labels L_{fw} and L_{bw}, either L_{fw} must represent a full 0-0′-path with $L_{fw}^{res} \leq HWP$ (so that L_{bw} represents the trivial path (0′)) or $L_{fw}^{res} > HWP$ is required.

Furthermore, if there exists a strictly monotone resource, the sequence of label propagation can be fully controlled with the help of buckets (for details we refer to Tilk *et al.*, 2017). However, none of our attributes propagated via (3.4) is strictly monotone. Hence, we consider the *divided demand* calculated by $d_j^{div} = d_{h(j)}/n_{h(j)}$ and introduce a new attribute L^{div}. For a route r that respects the soft-cluster constraints, the conditions $\sum_{h \in H(r)} d_h \leq Q$ is equivalent to $\sum_{i \in V(r)} d_i^{div} \leq Q$. Thus, the corresponding REF to propagate the divided demand from a label L_i ending at vertex i along arc (i, j) is $L_j^{div} = L_i^{div} + d_j^{div}$. Note that this attribute is strictly increasing and can therefore be used as the monotone resource.

The SoftCluVRP has a subproblem that is completely symmetric, i.e., any path is feasible if its reversed path is feasible, and vice versa. The consequence for the bidirectional labeling is therefore that only forward partial paths need to be generated and considered. The half-way point is set to $HWP = Q/2$ and we propagate forward labels L_i only if $L_i^{div} \leq HWP$. Any feasible backward partial path $p_{bw} = (i, j, \ldots, 0')$ exists as a feasible forward partial path $p_{fw} = (0, \ldots, j, i)$. For convenience, we define a *reversal operator* rev with $\text{rev}(p_{fw}) = p_{bw}$ and $\text{rev}(p_{bw}) = p_{fw}$, i.e., it reverses the partial path and exchanges 0 and 0′. In the final merge step, forward labels are then merged with other forward labels. Such an implicit bidirectional labeling has already been successfully implemented and applied by Bode and Irnich (2012) and Goeke *et al.* (2019).

Merge Procedure We now present the merge condition that tests whether two labels L and L' representing the forward paths p and p' from 0 to a customer vertex $i \in V \setminus \{0\}$ can be combined and produce a feasible 0-0′-path $r = (p, \text{rev}(p'))$. Considering the predecessor label of one of the two labels very much simplifies the merge condition. Thus, let L_- be the predecessor label of L, i.e., L results from the extension of L_- along an arc (j, i). The route r is feasible if

$$L_-^{load} + L'^{load} - \sum_{h \in H : L_-^{rem_h} \geq 0, L'^{rem_h} \geq 0} d_h \leq Q \tag{3.9a}$$

(which is equivalent to $L_-^{div} + L'^{div} \leq Q$)

$$\left\{ \begin{array}{rcll} L_-^{rem_h} & \leq & 0, & \text{if } L'^{rem_h} = -1 \\ L_-^{rem_h} & = & -1, & \text{if } L'^{rem_h} = 0 \\ L_-^{rem_h} + L'^{rem_h} & = & |V_h|, & \text{otherwise} \end{array} \right\} \qquad \forall h \in H \tag{3.9b}$$

$$L_-^{visit_v} + L'^{visit_v} \leq 1, \qquad \forall v \in V \tag{3.9c}$$

where (3.9a) checks the capacity constraint, (3.9b) that no cluster is served incompletely, and (3.9c) that no customer is visited more than once. (When storing unreachable instead of visited customers in the attributes L^{visit_v}, see Section 3.3.1.1, one direction must however rely on the actually visited customers. Moreover, for this direction, the original definition of the attributes L^{rem_h} must be used.) The cost of the resulting 0-0′-path $r = (p, \text{rev}(p'))$ is given by

$$\tilde{c}_r = L^{cost} + L'^{cost} + \sum_{h \in H: L^{rem_h} \geq 0, L'^{rem_h} \geq 0} \pi_h. \tag{3.10}$$

Note that the rightmost sum in (3.9a) and (3.10) considers the clusters that are incompletely served in both forward and backward direction. For these clusters, the demands d_h must be subtracted in (3.9a) and dual prices π_h must be added in (3.10) to prevent the incorrect double incorporation.

Dominance The following proposition shows that a bidirectional labeling algorithm is correct in two cases: (i) the weak dominance rule is applied or (ii) the strong dominance rule is applied in combination with a cost matrix that fulfills the *strict triangle inequality* (STI), i.e., for any three different vertices $i, j, k \in V$ the strict inequality $c_{ik} < c_{ij} + c_{jk}$ holds.

Proposition 1. *A bidirectional labeling algorithm that*
(i) either uses the weak dominance (Rule 1)
(ii) or the strong dominance (Rule 2) on a cost matrix that respects the STI
and the described merge procedure with conditions (3.9) *finds an optimal solution to the pricing subproblem of the SoftCluVRP.*

Proof. Let r be an optimal path, i.e., an elementary, feasible 0-0′-path with minimum reduced cost. We have to show that the bidirectional labeling algorithm either finds this path or one with the same reduced cost.

The forward labeling algorithm is certainly correct with either dominance, weak or strong, see Section 3.3.1.1. This has two consequences: First, all Pareto-optimal labels that arrive at the destination depot 0′ with divided demand $\leq HWP$ are found in the forward part of the bidirectional labeling. Hence, we can restrict the remainder of the proof to cases where all optimal routes r result from a merge.

With these assumptions, r can be represented as $r = (p, \text{rev}(q))$, where p and q both end at a merge vertex i.

Second, it can be assumed that $r = (p, \text{rev}(q))$ is generated by the complete forward labeling algorithm (otherwise there would exist another route fulfilling this). Therefore, the path p and its label L are considered in the merge of the bidirectional labeling algorithm. If the label L' of q would also be available in the merge, nothing remains to show.

Hence, we can now assume that L' is not generated, or it was generated and dominated. Then, there must exist a label L'^* that dominates L'. If $L'^* = L'$ (identical attributes), the path q^* associated with L'^* would produce the route $(p, \text{rev}(q^*))$ with identical reduced cost as r because of (3.10). A route equivalent to r is constructed.

Therefore, we can assume $L'^* \neq L'$. The proof now considers the two preconditions separately:

Case (i): We assume that the weak dominance (Rule 1) is applied. The weak dominance implies either $L'^{*cost} < L'^{cost}$ or that some clusters completely served with q are not touched by q^* (or both). Then, the pair L and L'^* also qualifies for the merge. The resulting 0-0′-path $(p, \text{rev}(q^*))$ has a reduced cost not greater than r because $L'^{*cost} \leq L'^{cost}$ and the (reverse) completion by p of both labels L' and L'^* produce the same sum of dual prices π_h in (3.4a). Hence, another route equivalent to r is constructed.

Case (ii): We assume that the strong dominance (Rule 2) is applied and that the cost matrix (c_{ij}) respects the STI. The two labels cannot differ only in cost, i.e., $L'^{*cost} < L'^{cost}$, because then the pair L and L'^* also qualifies for the merge and produces a route with smaller reduced cost than r, a contradiction!

Now, the only possibility left is that the two labels must also differ in the visit and remaining attributes. Recall that q^* and q are the partial paths associated with L'^* and L', respectively. As a backward path, p is a feasible (reverse) completion of q. As in the proof of Rule 2, there also exists a feasible completion p^* of q^* that results from path p by eliminating some vertices. (In our case, the different visit attributes ensure that at least one vertex is actually eliminated.) The STI now ensures that the (reduced) cost of this alternative completion p^* is strictly smaller than the (reduced) cost L^{cost} of p. As a consequence, the feasible route (p^*, q^*) has a strictly smaller reduced cost compared to $r = (p, q)$, again a contradiction! □

The use of the strong dominance (Rule 2) on a cost matrix that does not respect the STI but the TI may lead to incorrect results of the bidirectional labeling algorithm. This is shown in the following example.

Example 3. *We consider a SoftCluVRP instance with only one cluster* $V_1 = \{1, 2\}$ *with demand* $d_1 = 6$ *and vehicle capacity* $Q = 10$. *The divided demand of customers*

1 and 2 is $d_1^{div} = d_2^{div} = 3$ *and the half-way point is* $HWP = 5$*. The following cost matrix*

c_{ij}	$0'$	1	2
0	–	1	2
1	1	–	1
2	2	1	–

respects the TI but not the STI due to $c_{02} = c_{01} + c_{12}$*. Furthermore, we assume the dual prices to be* $\pi_1 = 2$ *and* $\mu = 4$*.*

The bidirectional labeling algorithm creates labels for the partial paths (0)*,* $(0, 1)$*,* $p = (0, 1, 2)$*, and* $p' = (0, 2)$*. However, no complete route is created with forward labeling due to the half-way point condition. Hence, routes can only result from a merge.*

The only dominance between labels occurs for p *and* p' *and the associated labels* $L = (L^{cost}, L^{load}, L^{rem_h}, L^{visit_v}) = (-2, 6, 0, (1, 1))$ *and* $L' = (L'^{cost}, L'^{load}, L'^{rem_h}, L'^{visit_v}) = (-2, 6, 1, (0, 1))$*, where* L *dominates* L' *using Rule 2. Thus, after applying a dominance algorithm,* L' *is discarded. The consequence is that no extension of* L' *is created either, in particular no label for the partial path* $(0, 2, 1)$*. Finally, the merge procedure does not find any feasible combination of labels.*

However, the route $r = (0, 1, 2, 0') = (p, rev(p'))$ *has cost* 4 *and reduced cost* $\tilde{c}_r = 4 - \pi_1 - \mu = -2 < 0$*. This shows that the bidirectional labeling with strong dominance may fail to produce optimal or even feasible solutions if the STI does not hold.*

3.3.1.3 Modification of the Cost Matrix

Some of the SoftCluVRP instances that we use in the later computational analysis do not respect the STI or even the TI. In order to apply bidirectional labeling algorithms, we transform these instances into equivalent new instances only differing in the cost matrix (c_{ij}) but not in optimal solutions. As every customer $k \in V \setminus \{0\}$ must be visited exactly once, it is possible to add any value $x = x(k) \in \mathbb{R}$ to all edges $\delta(k)$. The cost of all feasible solutions then increases by $2 \cdot x(k)$. This procedure is summarized as Procedure `Modify` that also records the modification in the customer-indexed array `modif`$[k]$.

If a SoftCluVRP instance does not respect the STI, we preprocess the instance with Algorithm 4. The first loop (Steps 1 to 5) resets the accumulated cost modification `modif`$[j]$ for each customer $j \in V \setminus \{0\}$ to zero, computes the largest violation of the TI when j is the middle vertex, and adds half of the maximum violation *vio* to the jth column and the jth row of the cost matrix. As we use

Procedure `Modify`

Input: A customer $k \in V \setminus \{0\}$, a value $x \in \mathbb{R}$
Output: Modified cost matrix (c_{ij}) and accumulated cost modification `modif`$[k]$

1 **for** $j \in V, j \neq k$ **do**
2 $\quad c_{jk} := c_{jk} + x$
3 $\quad c_{kj} := c_{kj} + x$
4 `modif`$[k] :=$ `modif`$[k] + 2x$

integer arithmetics for the routing costs $\lceil vio/2 \rceil + 1$ guarantees that the new matrix also comprises only integer values. After Step 5 the resulting cost matrix (c_{ij}) already respects the STI.

It can however happen that, for some customers $j \in V \setminus \{0\}$, the STI is fulfilled with a rather large slack. The strong dominance (see Rule 2) however benefits from a tightly fulfilled STI. Indeed, the chance to have a smaller reduced cost while having more customers visited increases with smaller routing costs. Therefore, the optional loop in Steps 7 to 13 iteratively decreases entries of the cost matrix as long as possible. We analyze the impact of this reduction (`reduce` = `true`) on the overall performance in the computational results section.

The final cost matrix (c_{ij}) computed by Algorithm 4 still respects the STI, the values `modif`$[j]$ give the overall modification for each customer j, and the cost of all feasible solutions increases by the constant $C := \sum_{j \in V \setminus \{0\}}$ `modif`$[j]$.

Example 4. (cont'd from Examples 1 and 2) *Recall that Example 1 gave an example where the strong dominance rule (Rule 2) was not applicable at vertex 2, because label L had higher reduced cost than the otherwise better label L'. Example 2 replaced the cost matrix (c_{ij}) of Example 1 by another cost matrix $(\dot{c}_{ij})$ so that the two labels' cost became identical and the strong dominance rule became applicable.*

The point is that matrix $(\dot{c}_{ij})$ results from matrix (c_{ij}) by subtracting -1 for each customer $j \in V \setminus \{0\}$. As a result, all edges $\{0, j\} \in \delta(0)$ have $\dot{c}_{0j} = c_{0j} - 1$, while the edges $\{i, j\} \in E \setminus \delta(0)$ connecting two customers i and j have $\dot{c}_{0j} = c_{0j} - 2$.

3.3.1.4 Heuristic Pricing and Acceleration Techniques

We now discuss three techniques that can be used to speed up the labeling algorithm: (1) the systematic violation of the triangle inequality, (2) decremental state space relaxation and *ng*-path relaxation, and (3) the use of reduced networks.

Systematic Violation of the Triangle Inequality The previous Section 3.3.1.3 has shown that a systematic modification (reduction) of the cost matrix

Algorithm 4: `Preprocessing( reduce )`

Input: Flag `reduce`
Output: Modified cost matrix (c_{ij}) and accumulated cost modification `modif`$[k]$

```
for j ∈ V \ {0} do
    modif[j] := 0
    vio := max_{i,k∈V,i≠j≠k,i≠k}(c_ik − c_ij − c_jk)
    if vio ≥ 0 then
        Modify(j, ⌈vio/2⌉ + 1)
update := reduce
while update do
    update := FALSE
    for j ∈ V \ {0} do
        slack := min_{i∈V,i≠j}{c_ij; min_{k∈V,i≠k≠j}(c_ij + c_jk − c_ik)}
        if slack > 2 then
            Modify(j, −⌈slack/2⌉ + 1)
            update := TRUE
```

can be used to create equivalent SoftCluVRP instances that have a higher chance to exploit the strong dominance rule (Rule 2). This chance can be further increased if we do not require the STI to be fulfilled after reducing the cost matrix. We determine the minimal entry $c_{min} = \min_{i,j \in V, i \neq j} c_{ij}$ in the cost matrix (c_{ij}) derived by Section 3.3.1.3 and calculate $offset = \lceil c_{min}/2 \rceil - 1$. Then, `Modify`$(j, -offset)$ is performed for each customer $j \in V \setminus \{0\}$. The resulting cost matrix $(\tilde{c}_{ij})$ possibly violates the (S)TI, but can be used together with strong dominance (Rule 2) as heuristic pricing.

Decremental State Space Relaxation and *ng*-Path Relaxation General elementary SPPRCs are NP-hard in the strong sense (Dror, 1994). The idea of a *decremental state space relaxation* (DSSR, Righini and Salani, 2008) is therefore to solve less difficult relaxations (less difficult in practice or in theory such as pseudo-polynomial relaxations). The relaxations must therefore be parametrizable so that a strongest relaxation guarantees elementary paths. One starts however with a weaker but relatively well-solvable SPPRC relaxation. If the solution of this relaxation contains a cycle, a stronger relaxation must be chosen and solved instead. The iterative process ends if an optimal solution to the relaxation is elementary (or just one with negative reduced costs).

We adopt the *ng*-path relaxation of Baldacci *et al.* (2011) to the SoftCluVRP. Let

the subset $N_i \subseteq V \setminus \{0\}$ be the vertex-specific *ng*-neighborhood of each vertex $i \in V$. The *ng*-path relaxation for $(N_i)_{i \in V}$ results from altering the REF (3.4d) of the visit attributes in the following way:

$$L_j^{visit_v} = \begin{cases} 0 & \text{if } v \notin N_j \\ L_i^{visit_v} + 1, & \text{if } v = j \text{ and } v = N_j \\ L_i^{visit_v}, & \text{if } v \neq j \text{ and } v = N_j \end{cases} \qquad \forall v \in V \setminus \{0\} \tag{3.11}$$

As a result, a feasible route may visit a customer more than once. However, the REF (3.4c) and conditions (3.5b) ensure that there are either exactly n_h visits to a cluster $V_h, h \in H$, or no visit at all.

There are several issues however related to the dominance rules introduced before. Already in monodirectional labeling (Section 3.3.1.1), the use of the strong dominance rule with a proper *ng*-relaxation can lead to undesirable results: A dominated label L' (Rule 2) can have feasible extensions that are however infeasible for the dominating label L.

Example 5. *We consider an example with two clusters* $V_1 = \{1, 2, 3\}$ *and* $V_2 = \{4, 5\}$. *Let* L_2 *be the label for partial path* $p = (0, 1, 3, 4, 2)$ *and* L_2' *for the partial path* $p' = (0, 4, 1, 2)$. *Since* p *has visited a superset of the customers compared to* p', *domination with Rule 2 is possible.*

Now consider the extension $q' = (2, 3, 4, 0')$ *of* p'. *It produces a non-elementary path* $r' = (p', q') = (0, 4, 1, 2, 3, 4, 0')$. *If* $N_1 = N_2 = N_4 = N_5 = V \setminus \{0\}$ *and* $N_3 = \{1, 2, 3, 5\}$, *then* r' *is a feasible ng-route in the sense of REFs* (3.11) *and conditions* (3.5).

However, the associated extension $q = (2, 4, 0')$ *of* p, *that result from the removal of the already visited customer* 3 *from* q', *produces the route* $r = (0, 1, 3, 4, 2, 4, 0')$ *that is infeasible w.r.t. the above ng-neighborhoods. In summary,* L_2' *is dominated but one of its extensions cannot be used to extend the dominating label* L_2. *If the route* r' *were optimal for the ng-path relaxation, the labeling algorithm with this strong dominance would not find* r'.

Such a behavior was first observed for VRPs with pickup-and-delivery (P&D) structure by Cherkesly *et al.* (2015). In essence, non-elementary paths can be incorrectly dominated and lower bounds computed with these relaxations are not unique. The bounds depend on the sequence of label extensions and dominance test. However, these bounds are valid as exploited in the selective pricing paradigm of Desaulniers *et al.* (2017).

As we want to use the *ng*-path relaxations in DSSR and bidirectional labeling, we will not use the strong dominance Rule 2. Note that intentionally the work of Gschwind *et al.* (2017) does not present an *ng*-path relaxation for a P&D-tailored bidirectional labeling algorithm with strong dominance, because the validity of this

combination is to date unclear. In line with these remarks, we now present another dominance rule, tailored for *ng*-path relaxations, for which correct domination can be shown for both monodirectional and bidirectional labeling.

Rule 3. *(ng-Dominance Rule) Let L and L' be two labels of different partial paths that end at the same vertex i. Label $L = (L^{cost}, L^{load}, L^{rem}, L^{visit})$ dominates label $L' = (L'^{cost}, L'^{load}, L'^{rem}, L'^{visit})$ if all of the following conditions are fulfilled:*

$$\text{(3.6a) } \textit{and } \text{(3.6b)} \tag{3.12a}$$

$$\left\{ \begin{array}{lll} (L^{rem_h} = L'^{rem_h} & \textit{and} & L_i^{visit_v} = L_i'^{visit_v}, \forall v \in V_h) \\ \textit{or } (L^{rem_h} \leq 0 & \textit{and} & L'^{rem_h} \geq 0) \end{array} \right\} \quad \forall h \in H \tag{3.12b}$$

Proof. Similar to Proof of Rule 2 and Proof of Proposition 1. □

Note that the *ng*-specific Rule 3 applies a stronger criterion than Rule 1 but a weaker criterion than Rule 2.

Heuristic/Partial Pricing using Reduced Networks Using reduced networks is a standard technique to speed-up the labeling algorithm (see, e.g., Dumas *et al.*, 1991). The elementary SPPRC (ESPPRC) is solved on an incomplete subgraph $\tilde{G} = (V', \tilde{A})$ with $\tilde{A} \subset A$. For the SoftCluVRP we characterize the subgraph by the non-negative integer parameter σ and define $\tilde{A}$ by

(i) all arcs connecting the origin or destination depot with customer vertices: $(i, j) \in A$ with $i = 0$ or $j = 0'$,
(ii) all intra-cluster arcs: $(i, j) \in A$ with $h(i) = h(j)$,
(iii) and σ inter-cluster arcs for every vertex $i \in V \setminus \{0\}$, connecting i to its σ nearest neighbors $j \notin V_{h(i)}$: $(i, j) \in A$ with $h(i) \neq h(j)$ and minimal routing costs c_{ij}.

A hierarchy of pricing heuristics can be used to iteratively solve the ESPPRC on subgraphs build by increasing parameter σ, until a route with negative reduced cost is found. Note that the complete graph G' must be used in the last iteration in order to find the exact solution (and possibly prove that no route with negative reduced cost exists).

3.3.1.5 Comparison with Pickup-and-Delivery Problems

We would like to point out that the SoftCluVRP shares some similarities with VRPs that have a pickup-and-delivery (P&D) structure. In the latter problems, the task is to fulfill a set of transportation requests, where each request i consists of the collection of some item(s) from a given pickup point i^+, the possibly shared transportation with other items, and the delivery of the item(s) to a given delivery point i^-. The basic observation is that clusters V_h of customers in the SoftCluVRP

SoftCluVRP	vs.	VRPs with P&D structure
Similarities:		
cluster V_h	$\sim$	request $\{i^+, i^-\}$
remaining customers	$\sim$	open requests
strong dominance based on subsets of remaining customers	$\sim$	strong dominance based on subsets of open requests
strong dominance and *ng*-path: incorrectly dominated non-elementary paths	$\sim$	strong dominance and *ng*-path: incorrectly dominated non-elementary paths (Cherkesly *et al.*, 2015)
Differences:		
no precedences	$\neq$	i^+ precedes i^-
dual prices for covering clusters are managed via REFs	$\neq$	dual prices for covering requests are incorporated into reduced cost matrix ($\tilde{c}_{ij}$)
strong dominance requires triangle inequality (TI) on original cost matrix (c_{ij})	$\neq$	strong dominance requires *delivery triangle inequality* (DTI) on reduced cost matrix ($\tilde{c}_{ij}$) (Ropke and Cordeau, 2009)
bidirectional labeling requires strict triangle inequality (STI)	$\neq$	bidirectional labeling requires two matrices (fw/bw with DTI/pickup TI) (Gschwind *et al.*, 2017)

Table 3.1: Similarities and differences between SoftCluVRP and VRPs with P&D structure, in particular regarding labeling algorithms for the SPPRC subproblem.

correspond to requests $i = \{i^+, i^-\}$ in P&D VRPs. More similarities but also the most important differences in these VRP variants and their SPPRC labeling subproblems (arising from their exact solution via column-generation approaches) are summarized in Table 3.1.

3.3.2 Branch-and-Cut

Our branch-and-cut algorithm for the SoftCluVRP subproblem is based on the formulation (3.3) and uses the callable library of CPLEX 12.8.1.0. for so-called lazy cuts and user cuts. We have kept all default settings of CPLEX except for enforcing CPLEX to run in single-thread mode. The initial linear program (LP) comprises the objective (3.3a), the depot degree constraints (3.3b), the coupling constraints (3.3c), and the capacity constraint (3.3e).

In the following we assume that all direct routes $r = (0, i, 0')$ for singleton clusters $V_h = \{i\}$ are already in the RMP. Therefore, these routes do not have to be priced out. The consequence is that we do not have to distinguish between binary and integer routing variables (see (3.3f) and (3.3g)), but all routing variables are binary.

For the detection of violated *subtour-elimination constraints* (SECs) we added callbacks to CPLEX. Let $(\bar{x}_e, \bar{z}_h)$ be a solution to the LP. We define the *support*

graph to $(\bar{x}_e, \bar{z}_h)$ as the graph $\bar{G} = (V, \bar{E})$ where the edge set is defined as $\bar{E} = \{e \in E : \bar{x}_e > 0\}$. The separation algorithm distinguishes between integer and fractional solutions.

For an integer solution $(\bar{x}_e, \bar{z}_h)$, we determine the connected components of $\bar{G}$. Our implementation uses an efficient implementation of a union-find algorithm (Tarjan, 1979). If a connected component induced by $S \subset V$ fulfills $0 \notin S$ and $|S| > 1$, violated SECs are found. Indeed, for all $i \in S$, the SEC to the pair (i, S) is violated (LHS is zero, RHS is two). We add only one SEC per subset S choosing i arbitrarily.

For fractional solutions $(\bar{x}_e, \bar{z}_h)$, we first also compute the connected components of $\bar{G}$. For a connected component induced by $S \subset V$ that fulfills $0 \notin S$ and $|S| > 1$, we determine $\bar{i} = \arg\max_{\bar{i} \in S} \bar{z}_{h(\bar{i})}$. If the SEC for the pair $(\bar{i}, S)$ is violated by more than a given threshold $\epsilon = 0.01$, we add the violated SEC. If none of the connected components with $0 \notin S$ gives a violated SEC, we analyze the connected component that contains the depot 0. Let $S_0 \subset V$ be the subset that induces this component. We next compute the maximum flow between the depot 0 and every vertex $i \in S_0, i \neq 0$ in the induced graph $G[S_0]$. These max-flow problems are solved with the algorithm of Boykov and Kolmogorov (2004) (available in the BOOST C++ library, `https://www.boost.org/doc/libs/1_67_0/libs/graph/doc/boykov_kolmogorov_max_flow.html`). Let the maximum-flow value be $\bar{f}_{0i}$. The degree of violation of the SEC for the (i, S_0) is $2\bar{z}_{h(i)} - \bar{f}_{0i}$. If several violations greater than ϵ exist, we choose $\bar{\imath}$ as one vertex that maximizes the violation and add the violated SEC for the pair $(\bar{\imath}, S_0)$ to the LP.

Note that we do not add separation procedures for other classes of valid inequalities such as, e.g., cover inequalities/cuts (Wolsey, 1998, p. 147f) induced by the knapsack-like constraints (3.3e) because these cuts are already implemented in CPLEX.

3.3.3 Primal Heuristic

We now present a metaheuristic for heuristic/partial pricing that systematically manipulates a given feasible route r using edge-exchange procedures for the TSP and additional operators that can drop the vertices of a cluster V_h with $h \in H(r)$ or add the vertices of a cluster V_h with $h \in H \setminus H(r)$ to the route r. The different operators are combined in variable neighborhood descent (VND) procedures (a variation of variable neighborhood search, see Mladenović and Hansen, 1997). We use the following basic operators:

`DoBest2Opt`(r): Search for a best-improving 2-OPT TSP move in r and perform this move if it is improving.

`DoBest3Opt`(r): The same, but with 3-OPT TSP moves.

`VND.TSP`(r): Perform a VND with the operators `DoBest2Opt` and `DoBest3Opt` on r.

`DropCluster`(r, h): Remove all vertices $i \in V_h$ from r.

`AddCluster`(r, h): Check whether the addition of the cluster V_h to r is feasible. If so, loop over all $i \in V_h$ and insert i into r at a position with smallest insertion cost. Otherwise leave r unchanged.

`DoBestDropCluster`(r): Loop over all $h \in H(r)$, make a copy r' of r, apply `DropCluster`(r', h) and `VND.TSP`(r'), and compute the reduced cost of the resulting route r''. Finally, set r to the route r'' with minimum reduced cost if $\tilde{c}_{r''} < \tilde{c}_r$, i.e., if r'' is improving. Otherwise leave r unchanged.

`DoBestAddCluster`(r): The same, but with a loop over all $h \in H \setminus H(r)$ and with `AddCluster` moves.

`VND`(r): Perform a VND with the operators `DoBestDropCluster` and `DoBestAddCluster` on r.

Our implementation of the best-improvement 2-OPT and 3-OPT local search procedures uses a so-called *radius* or *sequential search* mechanism to reduce the computational effort of local search for the quadratic and cubic neighborhoods, see (Bentley, 1992), (Hoos and Stützle, 2004, p. 373) and (Irnich *et al.*, 2006).

The starting point of the primal heuristic is the primal solution $\bar{y}_r$, $r \in \bar{\Omega}$ of the RMP (3.2). Note that all routes with $\bar{y}_r > 0$ have reduced cost $\tilde{c}_r = 0$, so that they are promising starting solutions. We loop over all these routes and apply the primal heuristic `PrimalHeuristic`(r), detailed in Algorithm 5, to each of them. In the primal heuristic, the loop (Steps 3 to 12) first tries to improve the current route r' by applying the `VND` with all operators, i.e., 2-OPT, 3-OPT, removal of a cluster, and addition of a cluster (in this order of increasing search effort). Then, it randomly removes up to three clusters from the resulting route r' (Steps 10 to 12). The loop is repeated up to 140 times, but a premature termination happens if the best found route r^* has negative reduced cost. All negative reduced-cost routes that are found are added to the RMP.

3.4 Branch-and-Price

Two important aspects of the branch-and-price algorithms for the SoftCluVRP are clarified now. In Section 3.4.1, we describe possible strategies for combining the heuristic pricing algorithms with a final exact labeling-based pricing algorithm. In Section 3.4.2, we discuss branching rules and their impact on pricing algorithms.

Algorithm 5: `PrimalHeuristic`(r) for the SoftCluVRP pricing subproblem

Input: A feasible route r
Output: A negative reduced-cost route r^* or `FAILED` if none is found

```
r* := r
r' := r
for Iter = 1, 2 ..., MaxIter do
    VND.TSP(r')
    VND(r')
    if c̃_{r'} < c̃_{r*} then
        r* := r'
    if c̃_{r*} < 0 then
        return r*
    for up to 3 times, as long as H(r') ≠ ∅ do
        Randomly choose h ∈ H(r')
        DropCluster(r', h)
return FAILED
```

3.4.1 Pricing Strategies

The different acceleration and heuristic pricing methods offer a plethora of pricing strategies. The question is how to combine the different pricing algorithms into a hierarchy of pricing algorithms (in the following referred to as *pricers*). If a pricer on a lower level of the hierarchy fails to produce (sufficiently good) negative reduced-cost routes, the pricer on the next level is called. In extensive preliminary experiments we tried numerous combinations. We summarize the most important findings of these experiments:

1. The use of the Preprocessing is beneficial (Algorithm 4 called with `reduce = TRUE`).
2. The primal heuristic pricer should be applied before the labeling pricers (Algorithm 5 of Section 3.3.3).
3. Bidirectional labeling (Section 3.3.1.2) very often outperforms monodirectional labeling (Section 3.3.1.1).
4. DSSR with *ng*-path relaxation is most of the time faster than directly solving the elementary SPPRC (Section 3.3.1.4).
5. Heuristic pricers that use the strong dominance rule should also systematically violate the TI to further accelerate computations (also Section 3.3.1.4).

Therefore, we create a hierarchy of nine pricers, where the first one is the primal heuristic pricer, the next seven are heuristic labeling algorithms, and the last one

is an exact labeling-based pricer. All labeling pricers use bidirectional labeling and DSSR. They differ in the size of the reduced networks (we use networks with 2, 5, 10 nearest neighbors, and the full network). For each network size, a first heuristic pricer applies the strong dominance Rule 2 to speed up computations even though this rule does not guarantee optimal solutions together with bidirectional labeling. With the same reasoning, the STI is systematically violated. The second pricer per network size is one that is an exact pricer for this network, i.e., it does apply the *ng*-specific dominance Rule 3 and does not further modify the distance after preprocessing.

Default Strategy				Alternative Strategies				
Level	Size of Reduced Network	Dominance Rule	Violate STI	w/o violation of STI	w/o reduction (`reduce = false`)	w/o DSSR + *ng*-path relax.	w/o primal heuristic	only mono-directional
0	full	—use primal heuristic—					×	
1	2	strong: Rule 2	yes	STI		×		mono
2	2	*ng*: Rule 3	no		×	×		mono, TI
3	5	strong: Rule 2	yes	STI		×		mono
4	5	*ng*: Rule 3	no		×	×		mono, TI
5	10	strong: Rule 2	yes	STI		×		mono
6	10	*ng*: Rule 3	no		×	×		mono, TI
7	full	strong: Rule 2	yes	STI		×		mono
8	full	*ng*: Rule 3	no		×	×		mono, TI

Table 3.2: Default pricing strategy and five alternative pricing strategies.

This default pricing strategy is depicted in the four leftmost columns of Table 3.2. In addition, we define five alternative pricing strategies that one-by-one vary one of the fundamental components of the default strategy. These alternative strategies are summarized in the five rightmost columns of Table 3.2.

In detail: (1) strategy 'w/o violation of STI' does not systematically modify the cost matrix so that the cost matrix fulfills the STI in every pricer, (2) strategy 'w/o reduction' calls the Preprocessing Algorithm 4 with `reduce = FALSE` so that the preprocessed cost matrix fulfills the STI with a larger slack, (3) strategy 'w/o DSSR + *ng*-path relax.' directly solves elementary subproblems instead of using DSSR, (4) strategy 'w/o primal heuristic' drops the pricer at level 0, and (5) strategy 'only monodirectional' replaces the bidirectional labeling by monodirectional labeling where the respective exact pricers reduce the cost matrix further so that only the TI and not the STI needs to hold.

In Section 3.5.2, we will compare the six strategies (default and five alternative) on SoftCluVRP benchmark instances. Moreover, by eliminating some pricers including level 8, i.e., the exact labeling on the full network, the pricing hierarchy can be complemented with the branch-and-cut pricer from Section 3.3.2. Results

with the respective pricing strategies are presented in Sections 3.5.3 and 3.5.4.

3.4.2 Branching

Note that in our model and the later analyzed benchmark instances the number of routes/vehicles is always given and fixed. Therefore, no branching on the number of vehicles is applicable here.

To finally ensure integrality of solutions, we apply the Ryan-Foster branching rule (Ryan and Foster, 1981) on the partitioning constraints (3.2b). These constraints ensure service for every cluster. Given an RMP solution $(\bar{y}_r)$, we determine for each pair $(h, h') \in H \times H, h < h'$ the number $f_{h,h'} = \sum_{r\in\Omega:a_{hr}=a_{h'r}=1} \bar{y}_r$. If the RMP solution is fractional, then there exists a value $f_{h,h'}$ strictly between 0 and 1 for some h and h'. We choose a pair with $f_{h,h'}$ closest to 0.5 and create the following two branches defined by the constraints

$$\sum_{r\in\Omega:a_{hr}=a_{h'r}=1} y_r = 0 \qquad \text{and} \qquad \sum_{r\in\Omega:a_{hr}=a_{h'r}=1} y_r = 1.$$

In the first branch, the *separate branch*, routes that serve both clusters V_h and $V_{h'}$ are not allowed. The second branch, the *together branch*, requires that the two clusters V_h and $V_{h'}$ are served by the same vehicle. Both types of branching decisions can be enforced without explicitly adding the above constraints to the RMP: Eliminate all routes $r \in \Omega$ that violate the branching condition from the current RMP. Moreover, ensure that forbidden routes are not priced out. This can be done as follows:

Separate Branch Separate branching decisions have an impact on the structure of the subproblem and likewise on the subproblem algorithms. Assume that two clusters V_h and $V_{h'}$ must be served separately. In the case of labeling (Section 3.3.1), we modify the propagation rule. Once that a first customer of V_h is visited, we do not extend to vertices of cluster $V_{h'}$. This new behavior can be achieved using binary attributes for all clusters. For the dominance, we do as if the attributes had values $L_j^{visit_v} = 1$ for all $v \in V_{h'}$ and $L_j^{rem_{h'}} = 0$, in order to mimic that cluster $V_{h'}$ were already served completely. For merge conditions, however, these attributes remain at their correct values $L_j^{visit_v} = 0$ for all $v \in V_{h'}$ and $L_j^{rem_{h'}} = -1$. The same is done with exchanged roles of h and h'.

In the case of branch-and-cut (Section 3.3.2), we add the constraint $z_h + z_{h'} \leq 1$. Nothing else has to be done.

In the case of the primal heuristic (Section 3.3.3), the operator `AddCluster`(r, h) must be modified. Recall that it adds cluster V_h to a given route r. If a separate constraint for h and h' is active, the operator adds V_h to route r but removes all vertices of cluster $V_{h'}$.

Together Branch A together branching decision for clusters V_h and $V_{h'}$ is trivial to impose for all types of subproblem algorithms. Instead of the original SoftCluVRP instance one just has to consider a new one in which the two clusters V_h and $V_{h'}$ are replaced by one bigger cluster $V_h \cup V_{h'}$ with demand $d_h + d_{h'}$.

3.5 Computational Results

Our algorithm is coded in C++ and compiled with MS Visual Studio 2015 in release mode. The callable library of CPLEX 12.8.1.0 is used to reoptimize the RMPs and to solve the subproblems with the branch-and-cut algorithm. We run all computations on a standard PC equipped with MS Windows 7 and an Intel(R) Core(TM) i7-5930K CPU clocked at 3.5 GHz and with 64 GB RAM of main memory. Our computer reaches a peak performance of 222.34 GFlops in the LINPACK benchmark (with 12 threads, evaluated with the program LinX 0.6.5).

3.5.1 SoftCluVRP Benchmark Instances

We test our algorithm on three benchmark sets. The first and second benchmark sets were derived from the CVRP benchmarks called `A`, `B`, `P`, `G`, and `C` by Bektaş *et al.* (2011). They defined θ as the desired average number of customers per customer cluster and, accordingly, $N = \lceil (n+1)/\theta \rceil$ customer clusters are built (for details, we refer to Fischetti *et al.*, 1997; Bektaş *et al.*, 2011). Choosing $\theta \in \{2, 3\}$, the `GVRP-2` and `GVRP-3` benchmarks comprise 79 instances each, resulting in 158 small- and medium-sized instances with 16 to 262 vertices and 6 to 131 clusters. The third set `Golden-Bat` was proposed by Battarra *et al.* (2014a) for the CluVRP and comprises 220 large-scale instances with 201 to 484 vertices and 14 to 97 clusters. They are based on the well-known CVRP instances by Golden *et al.* (1998). For each of the 20 original instances `Golden1` to `Golden20`, different clusterings were generated by choosing $\theta = \{5, \ldots, 15\}$, resulting in eleven groups with 20 instances each.

For all experiments reported in the following, we run our branch-and-price algorithms with a time limit of 1 hour (3600 seconds) per instance. All computation times are displayed in seconds.

3.5.2 Comparison of Labeling Strategies

In a first series of experiments, we analyze the performance of the six pricing strategies that we discussed in Section 3.4.1. To keep the computational effort limited, we restrict ourselves to solving the linear relaxation of the master program, i.e., the root node of the branch-and-bound tree. Moreover, we do not use all 158

benchmark instances but only those 32 instances that we were able to solve with some labeling-based pricing strategy during preliminary experiments.

Table 3.3 briefly summarizes the results obtained for the 32 instances. Detailed instance-by-instance results can be found in the respective Table 3.8 in Section 3.A of the Appendix. In both tables, T refers to the computation time for solving the linear relaxation of (3.2), *Avg.* T is the arithmetic mean, and *Geo.* T the geometric mean of the computation times over the 32 instances (the latter value is not so strongly affected by a few but large outliers). In Table 3.3, #Solved is the number of instances for which the linear relaxation could be solved within the time limit (of 1 hour).

	Time for solving the linear relaxation					
	default	w/o violation of STI	w/o reduction (`reduce = false`)	w/o DSSR + *ng*-path relax.	w/o primal heuristic	only mono-directional
Avg. T	903.9	1226.9	1569.7	2595.5	1518.9	1661.2
Geo. T	60.0	79.7	95.9	321.3	94.8	73.3
#Solved	30	27	22	10	23	21

Table 3.3: Comparison of labeling-based pricing strategies using 32 selected Soft-CluVRP instances.

The results for labeling-based pricing are very clear. From all acceleration techniques, the use of DSSR with the *ng*-path relaxation has the most positive impact: If replaced by directly solving subproblems as elementary SPPRC (strategy *w/o DSSR + ng-path relax.*), less than one third of the 32 linear relaxations can be solved. The impact of cost matrix reduction, primal heuristic, and bidirectional labeling is comparable, as only approximately 2/3 of the linear relaxations are solved when these techniques are not applied (see strategies *w/o reduction*, *w/o primal heuristic*, and *only monodirectional*, respectively). To use the systematic modification of the cost matrix (instead of strategy *w/o violation of STI*) leads to the smallest but still significant average speedups. The best strategy is clearly the *default* strategy in which all acceleration techniques switched on: 30 of the 32 linear relaxations are solved within the time limit, and average (arithmetic and geometric) computation times are considerably smaller compared to all other strategies.

3.5.3 Comparison of Pricing Strategies including Branch-and-Cut

In the second series of experiments, we want to find a best strategy for the use of the branch-and-cut-based subproblem algorithm introduced in Section 3.3.2. On the one hand, the primal heuristic can be called and in case of success it is not necessary to invoke the branch-and-cut algorithm. This is partial pricing with the primal heuristic. On the other hand, also the labeling-based pricing algorithms can be used for partial pricing. In this case, we use the default strategy from Section 3.4.1 but only the heuristic levels 1 and 3 (*truncated default strategy*). Moreover, the primal heuristic (level 0) may or may not be used. This leads to four possible strategies:

`Pure:` Pure branch-and-cut.
`+Prim:` Primal heuristic first.
`+Label:` Truncated default labeling strategy w/o primal heuristic first.
`+Prim+Label:` Truncated default labeling strategy with primal heuristic first.

For the comparison, we use the identical subset of 32 instances as in the previous section. The results are summarized in Table 3.4 and the corresponding instance-by-instance results are given in Table 3.8 in Section 3.A of the Appendix.

	Time for solving the linear relaxation			
	`Pure:`	`+Prim:`	`+Label:`	`+Prim+Label:`
primal heuristic:	no	yes	no	yes
truncated labeling:	no	no	yes	yes
Avg. T	7.9	2.6	152.7	9.9
Geo. T	2.5	0.9	5.7	1.9
#Solved	32	32	31	32

Table 3.4: Comparison of branch-and-cut-based pricing strategies using 32 selected SoftCluVRP instances.

The outcome of the experiments is that all four branch-and-cut-based pricing strategies outperform the labeling-based strategies by at least one order of magnitude of computation time. The worst strategy is `+Label` confirming that labeling-based pricing is inferior. Compared to the other branch-and-cut strategies, `+Label` fails in solving one linear relaxation (recall that the best labeling strategy failed in two cases). Moreover, the comparison shows that the primal heuristic is essential. The strategies `Pure` and `+Prim+Label` are incomparable w.r.t. arithmetic and geometric means of computation time. The best strategy (undominated) is `+Prim` where primal heuristic and branch-and-cut are combined. The speedup fac-

tor compared to the default labeling is >340 (ratio of arithmetic means) and >66 (ratio of geometric means).

For the remainder of the chapter, we use `+Prim` as the *default branch-and-cut strategy*.

3.5.4 Comparison of Labeling-based and Branch-and-Cut-based Pricing

Even if the previous sections clearly indicate the superiority of branch-and-cut over labeling for solving the subproblem, we want to test the two respective default strategies against each other on the full benchmark set and within the fully-fledged branch-and-price algorithm. Table 3.5 gives aggregated results for all 158 `GVRP` instances, grouped by the subclasses `A`, `B`, `P`, and `GC` of `GVRP-2` and `GVRP-3`. For the solution of the root, the table shows the number of successfully solved linear relaxation of (3.2) (#Solved) and average computation times T (*Avg.* and *Geo.*). Similarly, for the branch-and-price, $\#Int$ and $\#Opt$ is the number of instances where an integer solution is found and proven optimal, respectively.

	Default Labeling Strategy							Default Branch-and-Cut Strategy						
	linear relaxation			integer				linear relaxation			integer			
	#	Time T		#		Time T		#	Time T		#		Time T	
Set (#inst.)	Solved	*Avg.*	*Geo.*	Int	Opt	*Avg.*	*Geo.*	Solved	*Avg.*	*Geo.*	Int	Opt	*Avg.*	*Geo.*
`GVRP-2`														
`A (27)`	7	3000	2573	5	4	3353	3219	27	33	9	27	26	713	114
`B (23)`	7	2825	2229	3	2	3368	3198	23	24	10	23	17	1105	144
`P (24)`	11	2107	162	11	9	2393	250	24	155	4	23	23	474	17
`GC (5)`	0	3600	3600	0	0	3600	3600	1	2912	1929	1	1	2994	2489
`GVRP-3`														
`A (27)`	0	3600	3600	0	0	3600	3600	27	17	5	27	27	83	17
`B (23)`	1	3522	3494	0	0	3600	3600	23	7	4	23	23	160	18
`P (24)`	8	2411	316	8	8	2431	367	24	32	2	24	24	107	4
`GC (5)`	0	3600	3600	0	0	3600	3600	1	2899	1747	1	1	3221	3101
Total (158)	34	2966	1362	27	23	3163	1637	150	225	7	149	142	605	36

Table 3.5: Results for the 158 `GVRP` instances.

Branch-and-price with the default labeling strategy cannot solve any group of instances completely, neither the linear relaxation nor the full branch-and-bound tree. It seems however that instances from the group `GVRP-2` are slightly easier for this approach than from the group `GVRP-3`. In comparison, branch-and-price with the default branch-and-cut strategy performs much better than the default labeling strategy: 34 vs. 150 solved linear relaxations and 23 vs. 142 exactly solved instances, respectively. The observed aggregated computation times underline the

impressive predominance of branch-and-cut for the SoftCluVRP subproblems. We would like to add that the labeling-based approach is almost always inferior in a per instance comparison (however, six instances all with computation times below 2 seconds are solved faster with labeling).

Detailed instance-by-instance results of the branch-and-price with the default branch-and-cut strategy are given in Tables 3.9 to 3.12 in Section 3.B of the Appendix.

3.5.5 Results for the `Golden-Bat` Instances

Since the `Golden-Bat` instances are much larger regarding the number of customers and the number of clusters, they are not at all accessible for branch-and-price algorithms that use the labeling techniques of Section 3.3.1. In this section, we therefore restrict the presentation of results to those obtained with the branch-and-price that uses the default branch-and-cut strategy.

	linear relaxation			integer			
	#	Time T		#		Time T	
$n+1$	Solved	*Avg.*	*Geo.*	Int	Opt	*Avg.*	*Geo.*
201 (11)	10	832	487	10	10	832	487
241 (22)	17	1553	963	11	9	2307	1426
253 (11)	9	938	394	9	9	1119	475
256 (11)	10	943	494	8	8	1446	747
281 (11)	5	2702	2392	5	5	2702	2392
301 (11)	6	2338	1918	6	6	2338	1918
321 (22)	7	2875	2485	5	5	3196	2950
324 (11)	6	2179	1483	6	6	2453	2140
361 (22)	5	3171	2936	5	5	3171	2936
397 (11)	2	3256	3142	1	1	3561	3559
400 (11)	3	3112	2945	1	1	3515	3502
401 (11)	0	3600	3600	0	0	3600	3600
421 (11)	3	3033	2790	3	3	3033	2790
441 (11)	0	3600	3600	0	0	3600	3600
481 (22)	0	3600	3600	0	0	3600	3600
484 (11)	0	3600	3600	0	0	3600	3600
Total (220)	83	2626	1927	70	68	2817	2172

(a) Grouped by number of vertices $n+1$.

	linear relaxation			integer			
	#	Time T		#		Time T	
θ	Solved	*Avg.*	*Geo.*	Int	Opt	*Avg.*	*Geo.*
5 (20)	1	3521	3497	1	1	3521	3497
6 (20)	1	3523	3501	0	0	3600	3600
7 (20)	4	3136	2891	2	2	3394	3261
8 (20)	5	3036	2568	4	4	3042	2572
9 (20)	6	2866	2308	5	5	2988	2582
10 (20)	9	2603	1939	7	7	2816	2169
11 (20)	9	2496	1726	7	7	2744	1937
12 (20)	9	2326	1562	8	8	2617	1898
13 (20)	13	1889	1139	13	12	2140	1340
14 (20)	14	1705	1085	12	11	2097	1382
15 (20)	12	1790	1003	11	11	2031	1258
Total (220)	83	2626	1927	70	68	2817	2172

(b) Grouped by average cluster size θ.

Table 3.6: Results for the 220 large-scale `Golden-Bat` instances.

Tables 3.6a and 3.6b summarize the results, where the first table groups the instances by the number $n+1$ of vertices and the second by the average cluster size θ. On the one hand, the difficulty of instances increases with the number of vertices, which seems natural. The three largest instances solved have 420 customers. On the other hand, instances with a larger average cluster size θ become better solvable. Also this behaviour is expected because with larger clusters the

number of partitioning constraints (3.2b) in the master program decreases and also the model (3.3) of the subproblem has less cluster variables z_h making it less difficult to solve. The smallest average cluster size of the above solved 420-customer instances has $\theta = 13$. Detailed instance-by-instance results are reported in Tables 3.13 to 3.16 in Section 3.C of the Appendix.

3.5.6 Comparison of CluVRP and SoftCluVRP Solutions

For those instances, for which optimal SoftCluVRP and CluVRP solutions are known, we compute the *gap to CluVRP* (in percent) as $100 \cdot (Z_{CluVRP} - Z_{SoftCluVRP})/Z_{SoftCluVRP}$ where the minimal routing costs are $Z_{SoftCluVRP}$ and Z_{CluVRP}, respectively. Table 3.7 shows aggregated results for the `GVRP-3` and `Golden-Bat` instances. Note that detailed results for the benchmark `GVRP-2` have not been published so that for them Z_{CluVRP} is not known. We took the detailed results for the benchmark `GVRP-3` from (Defryn and Sörensen, 2017).

θ	Set (#inst.)	#both opt	Avg. gap to CluVRP
3			
	A (27)	27	2.66
	B (23)	23	1.16
	P (24)	24	4.73
	GC (5)	2	3.73
Total	(79)	76	2.89

(a) `GVRP-3` instances.

θ (#inst.)	#both opt	Avg. gap to CluVRP
5 (20)	1	9.45
6 (20)	3	7.21
7 (20)	4	7.46
8 (20)	6	6.79
9 (20)	7	7.46
10 (20)	8	6.90
11 (20)	9	6.16
12 (20)	11	5.94
13 (20)	12	5.62
14 (20)	13	5.51
15 (20)	13	5.46
Total (220)	87	6.21

(b) `Golden-Bat` instances.

Table 3.7: Comparison of optimal SoftCluVRP and CluVRP solutions.

The results for the `GVRP-3` instances in Table 3.7a show for how many instances optimal solutions of both SoftCluVRP and CluVRP are known (#both opt). Over these 76 instances, the average gap varies over the subsets `A`, `B`, `P`, and `GC`. Gaps are larger in subset `P` compared to `A`, and these larger compared to `B`. We attribute this difference to the different ways in which the subsets were constructed.

The results for the `Golden-Bat` instances are shown in Table 3.7b: Average gaps strongly depend on the average number θ of customers per cluster. The

larger $\theta \geq 5$, the smaller the average gaps. This behaviour is intuitive, since a SoftCluVRP route tends to change clusters less frequently when clusters are larger. Thus, serving the same clusters but respecting hard-cluster constraints requires only fewer modifications on the SoftCluVRP route. If θ grows even further so that routes serve single clusters, the gap vanishes.

The described dependency of the gaps on θ is definitely not the same for smaller values $\theta \leq 5$: It is impossible that gaps constantly increase when θ decreases, because the extreme case of $\theta = 1$ means that clusters are singleton sets, for which optimal SoftCluVRP and CluVRP coincide again. Hence, gaps are zero for $\theta = 1$. This is in line with what the comparison of the `GVRP-3` and `Golden-Bat` benchmarks shows: The former benchmark has $\theta = 3$ and smaller average gaps compared to the larger gaps for $\theta \geq 5$ for the latter benchmark.

3.6 Conclusions

In this chapter, we have designed and analyzed different branch-and-price algorithms for the exact solution of the SoftCluVRP. The research has mainly focussed on the solution of the column-generation subproblem, a variant of the SPPRC. It has turned out that, for this variant of the vehicle-routing problem, dynamic programming-based labeling algorithms are strongly outperformed by a branch-and-cut algorithm that works directly on the SoftCluVRP subproblem formulation. The latter integer programming-based solution approach contributes with computation times that are by one order of magnitude shorter than those of sophisticated dynamic-programming labeling algorithms. The largest SoftCluVRP instances that we solved to optimality have more than 400 customers or more than 50 clusters.

We attribute the success of branch-and-cut for the solution of SoftCluVRP subproblems to the following facts: The subproblem is very close to a TSP, more precisely, it is a TSP with profits that also shares characteristics with the prize-collecting TSP (Feillet *et al.*, 2005; Balas, 1989). For these types of TSPs, branch-and-cut is the leading state-of-the-art solution approach (Gutin and Punnen, 2007). Previous attempts of using IP-based methods for SPPRCs, like (Jepsen *et al.*, 2008), concentrated on the solution of CVRP subproblems via branch-and-cut. Results were competitive with monodirectional labeling algorithms without DSSR of that time. However, powerful techniques such as bidirectional labeling and the *ng*-path relaxation to be used separately or within DSSR were not yet available when Jepsen *et al.* (2008) conducted their experiments. When including these newer techniques, labeling algorithms outperform the branch-and-cut algorithm on CVRP subproblems.

In comparison, the SoftCluVRP subproblem is even closer to a TSP than the

CVRP subproblem. The point is that in the CVRP subproblem visits are decided on a customer level, while in the SoftCluVRP subproblem on a cluster level, and nothing is to decide regarding visits in a pure TSP. This may explain why branch-and-cut is superior to fully-fledged labeling algorithms in SoftCluVRP subproblems but not in CVRP subproblems.

Future research may focus on further enhancing the branch-and-cut with new classes of valid inequalities as well as effective and fast separation heuristics. We think that soft-cluster constraints could also play an important role in districting and capacitated arc-routing applications (Butsch *et al.*, 2014; Belenguer *et al.*, 2014).

Acknowledgment

This research was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant IR 122/10-1. This support is gratefully acknowledged.

Bibliography

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, **19**(6), 621–636.

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

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

Battarra, M., Erdoğan, G., and Vigo, D. (2014a). 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.

Belenguer, J. M., Benavent, E., and Irnich, S. (2014). The capacitated arc routing problem: Exact algorithms. In *Arc Routing*, chapter 9, pages 183–221. Society for Industrial & Applied Mathematics (SIAM).

Bentley, J. J. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, **4**(4), 387–411.

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

Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **26**, 1124–1137.

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

Cherkesly, M., Desaulniers, G., and Laporte, G. (2015). Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading. *Transportation Science*, **49**(4), 752–766.

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

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desaulniers, G., Pecin, D., and Contardo, C. (2017). Selective pricing in branch-price-and-cut algorithms for vehicle routing. *EURO Journal on Transportation and Logistics*, **8**(2), 147–168.

Drexl, M. and Irnich, S. (2012). Solving elementary shortest-path problems as mixed-integer programs. *OR Spectrum*, **36**(2), 281–296.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Reasearch*, **42**(5), 977–978.

Dumas, Y., Desrosiers, J., and Soumis, F. (1991). The pick-up and delivery problem with time windows. *European Journal of Operational Research*, **54**, 7–22.

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

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

Feillet, D., Dejax, P., Gendreau, M., and Guéguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Feillet, D., Dejax, P., and Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science*, **39**(2), 188–205.

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.

Goeke, D., Gschwind, T., and Schneider, M. (2019). Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier. *Discrete Applied Mathematics*, **264**, 43–61.

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

Gschwind, T., Irnich, S., Rothenbächer, A.-K., and Tilk, C. (2017). Bidirectional labeling in column-generation algorithms for pickup-and-delivery problems. *European Journal of Operational Research*, **266**, 521–530.

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

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann.

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

Irnich, S. and Villeneuve, D. (2006). The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, **18**(3), 391–406.

Irnich, S., Funke, B., and Grünert, T. (2006). Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, **33**(8), 2405–2429.

Irnich, S., Toth, P., and Vigo, D. (2014). The family of vehicle routing problems. In P. Toth and D. Vigo, editors, *Vehicle Routing*, chapter 1, pages 1–33. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Jepsen, M. K., Petersen, B., and Spoorendonk, S. (2008). A branch-and-cut algorithm for the elementary shortest path problem with a capacity constraint. Technical report 08-01, Department of Computer Science, University of Copenhagen, Kopenhagen, Denmark.

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

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

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

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

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, **51**(3), 155–170.

Ropke, S. and Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, **43**(3), 267–286.

Ryan, D. and Foster, B. (1981). An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, chapter 17, pages 269–280. Elsevier, North-Holland.

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.

Tarjan, R. E. (1979). A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, **18**(2), 110–127.

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

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

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

Wolsey, L. A. (1998). *Integer Programming*. Wiley-Interscience, New York, NY.

Appendix

3.A Linear-Relaxation Results for the 32 Filtered GVRP Instances

Table 3.8 shows detailed instance-by-instance results for the 32 filtered GVRP instances per labeling- and branch-and-cut-strategy as described in Sections 3.5.2 and 3.5.3.

Instance						Labeling-Strategies						Branch-&-Cut-Strategies			
θ	Set	n	k	N	m	default	w/o violation of STI	w/o reduction (`reduce = false`)	w/o DSSR + ng-path relax.	w/o primal heuristic	only mono-directional	`Pure`	`+Prim`	`+Label`	`+Prim +Label`
2	A	32	5	17	3	193.3	311.2	163.4	3600.0	116.8	127.0	3.6	1.1	2.1	1.5
	A	32	6	17	3	1268.5	3164.2	3394.2	3600.0	2220.2	2368.4	3.8	2.9	9.6	4.5
	A	33	5	17	3	984.5	2302.1	1728.0	3600.0	2672.3	3600.0	4.3	1.2	14.9	5.3
	A	36	6	19	3	3596.3	3600.0	1709.4	3600.0	3600.0	3600.0	8.2	4.0	93.1	32.1
	A	37	5	19	3	942.4	3600.0	1195.6	3600.0	1382.5	3600.0	6.7	1.4	10.5	4.4
	A	44	6	23	4	1159.4	3370.9	3600.0	3600.0	3440.2	3567.0	7.8	4.1	8.3	4.4
	A	54	9	28	5	1053.0	1100.1	3600.0	3600.0	3600.0	3600.0	28.2	3.8	232.3	45.1
	B	30	5	16	3	132.3	145.5	92.3	3600.0	210.1	129.3	3.0	1.1	6.0	5.0
	B	34	5	18	3	1599.7	1671.4	3600.0	3600.0	3600.0	3600.0	7.3	2.8	5.1	2.1
	B	37	6	19	3	531.1	576.7	1587.0	3600.0	557.6	3600.0	7.8	1.6	5.3	1.4
	B	43	7	22	4	2639.6	2827.9	3464.0	3600.0	3600.0	3600.0	12.7	3.3	15.1	5.5
	B	44	5	23	3	957.7	1022.2	3600.0	3600.0	3600.0	3600.0	14.9	4.8	33.6	5.2
	B	49	7	25	4	756.7	815.8	3600.0	3600.0	3600.0	176.1	12.3	1.3	38.3	3.8
	B	50	7	26	4	463.8	504.9	3600.0	3600.0	1652.5	3600.0	29.8	9.7	161.3	51.5
	P	15	8	8	5	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	0.1	0.1	0.1	0.1
	P	18	2	10	2	0.2	0.8	1.3	3.9	0.7	0.5	0.2	0.1	0.1	<0.1
	P	19	2	10	2	4.8	10.5	14.9	679.3	6.7	4.7	0.2	<0.1	0.1	0.1
	P	20	2	11	2	9.4	11.2	27.6	982.7	7.4	1.6	0.2	0.1	0.1	0.1
	P	21	2	11	2	15.6	26.0	42.9	3600.0	42.4	17.3	0.1	0.1	0.5	0.1
	P	21	8	11	5	0.1	<0.1	<0.1	<0.1	0.1	<0.1	0.1	0.1	0.2	0.2
	P	22	8	12	5	0.1	<0.1	<0.1	0.1	<0.1	<0.1	0.6	0.3	0.3	0.2
	P	49	10	25	5	3600.0	3600.0	3600.0	3600.0	3600.0	3257.1	19.2	11.2	64.7	58.2
	P	50	10	26	6	369.4	1225.1	488.6	3600.0	1117.6	328.3	10.1	2.9	26.0	15.4
	P	54	10	28	5	3117.1	3600.0	3600.0	3600.0	3600.0	3452.9	26.2	4.7	136.9	11.1
	P	54	15	28	8	47.9	58.9	42.8	2188.7	59.1	16.0	20.2	12.9	19.9	7.8
	P	59	15	30	8	105.5	192.7	240.1	3600.0	239.4	74.6	14.0	6.3	16.1	10.3
3	B	30	5	11	2	1708.0	1841.8	3601.1	3600.0	2336.2	3600.0	2.0	0.5	370.2	8.9
	B	38	5	13	2	3600.0	3600.0	3600.0	3600.0	3600.0	3600.0	2.5	0.2	3600.0	23.9
	P	15	8	6	4	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
	P	21	8	8	4	<0.1	0.1	0.1	0.1	<0.1	<0.1	0.1	0.2	0.2	0.2
	P	22	8	8	3	0.1	0.2	0.1	0.2	0.3	0.1	0.5	0.1	0.6	0.2
	P	54	15	19	6	68.1	80.8	38.9	3600.0	141.6	37.1	4.7	1.6	13.6	7.2
Avg. T						903.9	1226.9	1569.7	2595.5	1518.9	1661.2	7.9	2.6	152.7	9.9
Geo. T						60.0	79.7	95.9	321.3	94.8	73.3	2.5	0.9	5.7	1.9

Table 3.8: Detailed results for 32 filtered `GVRP` instances: Time T (in seconds) per labeling- and branch-and-cut-strategy for solving the linear relaxation.

3.B Detailed Results for the GVRP Instances

Detailed instance-by-instance results of the branch-and-price with the default branch-and-cut strategy are given in Tables 3.9 to 3.12. We describe the instance (number of customers n, number of vehicles k in the original CVRP instance, number of customer clusters N, number of vehicles m, and average cluster size θ) and additionally provide the following information:

BKS: Best known solution, bold if proven optimal;
Gap to CluVRP: Gap in percent between optimal CluVRP and SoftCluVRP solutions, see Section 3.5.6;
First found by: Article that first computed the BKS (Bat14=Battarra *et al.* (2014a), DS17=Defryn and Sörensen (2017), Vidal15=Vidal *et al.* (2015), B&P=this work);
LB root: Lower bound provided by the linear relaxation;
LB tree: Lower bound provided by branch-and-price;
UB: Upper bound provided by branch-and-price;
#B&B nodes: Number of branch-and-bound nodes explored;
Time T: Computation time (in seconds).

Furthermore, for the column *First found by* and our branch-and-price-algorithm we give the following information:

B&P: this work with the default branch-and-cut pricing (`+Prim`);
B&P*: found during computational studies with another variant ($\neq$ `+Prim`);
B&P$^{\diamondsuit}$: as B&P*, but with an extended time limit (7200 seconds);
B&P†: as B&P, but with an extended time limit (up to 36,000 seconds).

Regarding BKS and optimal solutions, recall that the SoftCluVRP is a relaxation of the CluVRP. Any lower bound for the SoftCluVRP is also a lower bound for the respective CluVRP. The same holds for upper bounds of the CluVRP that are upper bounds for the respective SoftCluVRP. Therefore, the heuristics of Defryn and Sörensen (2017) and Vidal *et al.* (2015) as well as the exact approach for the CluVRP by Battarra *et al.* (2014a) provide upper bounds that may be BKS or may even prove optimality of our solutions when the branch-and-price is stopped at the time limit. Note that here, in all cases the optimal solution is known, the proof of optimality is given by our branch-and-price algorithm, also if this solution was found by Defryn and Sörensen (2017), Vidal *et al.* (2015), or Battarra *et al.* (2014a) before.

Instance								Branch-and-Price results				
	n	k	N	m	BKS	*Gap to CluVRP*	*First found by*	LB root	LB tree	UB	#B&B nodes	Time T
A	31	5	16	2	**595**	-	B&P	595	595	595	1	7
A	32	5	17	3	**528**	-	B&P	517	528	528	17	12
A	32	6	17	3	**561**	-	B&P	560	561	561	4	5
A	33	5	17	3	**568**	-	B&P	562	568	568	14	13
A	35	5	18	2	**596**	-	B&P	589	596	596	12	65
A	36	5	19	3	**573**	-	B&P	571	573	573	5	6
A	36	6	19	3	**660**	-	B&P	660	660	660	1	4
A	37	5	19	3	**547**	-	B&P	547	547	547	1	1
A	38	5	20	3	**659**	-	B&P	639	659	659	37	78
A	38	6	20	3	**676**	-	B&P	658	676	676	41	78
A	43	6	22	3	**723**	-	B&P	722	723	723	3	23
A	44	6	23	4	**679**	-	B&P	679	679	679	1	4
A	44	7	23	4	**774**	-	B&P	761	774	774	60	242
A	45	7	23	4	**708**	-	B&P	685	708	708	143	209
A	47	7	24	4	**784**	-	B&P	760	784	784	356	1431
A	52	7	27	4	**732**	-	B&P	707	732	732	68	285
A	53	7	27	4	**806**	-	B&P	797	806	806	26	265
A	54	9	28	5	**778**	-	B&P	765	778	778	25	84
A	59	9	30	5	**877**	-	B&P	860	877	877	133	2010
A	60	9	31	5	**749**	-	B&P	744	749	749	14	142
A	61	8	31	4	**849**	-	B&P	838	849	849	34	839
A	62	9	32	5	**1043**	-	B&P	1030	1043	1043	123	3159
A	62	10	32	5	**895**	-	B&P	882	895	895	55	512
A	63	9	32	5	**895**	-	B&P	886	895	895	41	1132
A	64	9	33	5	**825**	-	B&P	800	825	825	313	2544
A	68	9	35	5	**857**	-	B&P	838	857	857	337	2506
A	79	10	40	5	**1115**	-	B&P	1110	1113	1115	30	3600
B	30	5	16	3	**451**	-	B&P	449	451	451	7	7
B	33	5	17	3	**495**	-	B&P	484	495	495	17	26
B	34	5	18	3	**654**	-	B&P	613	654	654	23	27
B	37	6	19	3	**479**	-	B&P	479	479	479	2	3
B	38	5	20	3	**378**	-	B&P	372	378	378	3	5
B	40	6	21	3	**514**	-	B&P	509	514	514	3	13
B	42	6	22	3	**522**	-	B&P	498	522	522	320	897
B	43	7	22	4	**562**	-	B&P	531	562	562	183	363
B	44	5	23	3	**542**	-	B&P	542	542	542	2	7
B	44	6	23	4	**506**	-	B&P	491	506	506	73	141
B	49	7	25	4	**495**	-	B&P	495	495	495	1	1
B	49	8	25	5	954	-	B&P$^{\diamond}$	904	938	958	981	3600
B	50	7	26	4	**672**	-	B&P	661	672	672	37	123
B	51	7	26	4	**485**	-	B&P	471	485	485	57	224
B	55	7	28	4	520	-	B&P	457	476	520	972	3600
B	56	7	29	4	777	-	B&P*	716	744	781	317	3600
B	56	9	29	5	**983**	-	B&P	980	983	983	33	251
B	62	10	32	5	**865**	-	B&P	850	865	865	50	1402
B	63	9	32	5	**550**	-	B&P	550	550	550	1	21
B	65	9	33	5	850	-	B&P$^{\diamond}$	827	836	869	186	3600
B	66	10	34	5	725	-	B&P$^{\diamond}$	682	699	736	587	3600
B	67	9	34	5	**745**	-	B&P	744	745	745	8	293
B	77	10	39	5	**842**	-	B&P$^{\diamond}$	815	832	849	130	3600

Table 3.9: Detailed results for the `GVRP-2` instances, subsets `A` and `B`.

Instance								Branch-and-Price results				
	n	k	N	m	BKS	*Gap to CluVRP*	*First found by*	LB root	LB tree	UB	#B&B nodes	Time T
P	15	8	8	5	**299**	-	B&P	299	299	299	1	<1
P	18	2	10	2	**195**	-	B&P	195	195	195	1	<1
P	19	2	10	2	**208**	-	B&P	208	208	208	2	<1
P	20	2	11	2	**208**	-	B&P	208	208	208	1	<1
P	21	2	11	2	**209**	-	B&P	209	209	209	1	<1
P	21	8	11	5	**397**	-	B&P	397	397	397	1	<1
P	22	8	12	5	**369**	-	B&P	366	369	369	8	2
P	39	5	20	3	**401**	-	B&P	400	401	401	7	10
P	44	5	23	3	**443**	-	B&P	443	443	443	1	5
P	49	7	25	4	**464**	-	B&P	458	464	464	46	119
P	49	8	25	4	**501**	-	B&P	490	501	501	24	230
P	49	10	25	5	**512**	-	B&P	509	512	512	20	56
P	50	10	26	6	**548**	-	B&P	545	548	548	13	21
P	54	7	28	4	**477**	-	B&P	477	477	477	1	8
P	54	8	28	4	**484**	-	B&P	480	484	484	8	40
P	54	10	28	5	**514**	-	B&P	514	514	514	1	5
P	54	15	28	8	**684**	-	B&P	681	684	684	18	70
P	59	10	30	5	**575**	-	B&P	564	575	575	84	725
P	59	15	30	8	**700**	-	B&P	692	700	700	118	216
P	64	10	33	5	**616**	-	B&P	611	616	616	28	291
P	69	10	35	5	**643**	-	B&P	635	643	643	71	934
P	75	4	38	2	**557**	-	B&P	555	557	557	34	2745
P	75	5	38	3	**571**	-	B&P	567	571	571	49	2311
P	100	4	51	2	646	-	B&P$^{\diamond}$	645	645	-	1	3600
G	261	25	131	12	3693	-	Vidal15	-	-	-	0	3600
C	100	10	51	5	**628**	2.18	B&P	626	628	628	4	569
C	120	7	61	4	807	-	Bat14	-	-	-	0	3600
C	150	12	76	6	816	-	Bat14	-	-	-	0	3600
C	199	16	100	8	955	-	Bat14	-	-	-	0	3600

Table 3.10: Detailed results for the `GVRP-2` instances, subsets `P` and `GC`.

Instance								Branch-and-Price results				
	n	k	N	m	BKS	*Gap to CluVRP*	*First found by*	LB root	LB tree	UB	#B&B nodes	Time T
A	31	5	11	2	**515**	1.34	DS17	515	515	515	1	<1
A	32	5	11	2	**461**	2.33	DS17	461	461	461	1	<1
A	32	6	11	2	**554**	1.42	DS17	551	554	554	3	2
A	33	5	12	2	**538**	1.65	DS17	529	538	538	11	6
A	35	5	12	2	**543**	7.65	DS17	535	543	543	8	6
A	36	5	13	2	**545**	4.22	B&P	545	545	545	16	11
A	36	6	13	2	**605**	1.63	DS17	603	605	605	3	6
A	37	5	13	2	**507**	0.00	Bat14	500	507	507	4	5
A	38	5	13	2	**588**	3.61	DS17	568	588	588	15	13
A	38	6	13	2	**603**	1.63	DS17	595	603	603	7	6
A	43	6	15	2	**691**	3.22	DS17	686	691	691	3	27
A	44	6	15	3	**652**	8.43	DS17	652	652	652	1	2
A	44	7	15	3	**661**	0.45	DS17	651	661	661	9	29
A	45	7	16	3	**642**	3.31	DS17	641	642	642	5	12
A	47	7	16	3	**680**	0.44	DS17	674	680	680	28	57
A	52	7	18	3	**627**	3.69	DS17	625	627	627	3	14
A	53	7	18	3	**699**	3.45	DS17	679	699	699	51	143
A	54	9	19	3	**645**	1.23	DS17	645	645	645	1	9
A	59	9	20	3	**762**	3.18	DS17	761	762	762	3	29
A	60	9	21	4	**671**	1.61	DS17	671	671	671	3	23
A	61	8	21	3	**771**	0.9	DS17	761	771	771	36	404
A	62	10	21	4	**779**	2.75	DS17	779	779	779	1	13
A	62	9	21	3	**837**	3.24	DS17	837	837	837	1	43
A	63	9	22	3	**767**	0.78	DS17	754	767	767	54	585
A	64	9	22	3	**693**	4.41	DS17	693	693	693	1	14
A	68	9	23	3	**794**	2.46	DS17	779	794	794	31	603
A	79	10	27	4	**944**	2.88	DS17	944	944	944	1	178
B	30	5	11	2	**375**	0.00	Bat14	366	375	375	13	4
B	33	5	12	2	**415**	0.24	DS17	397	415	415	19	5
B	34	5	12	2	**557**	0.89	DS17	526	557	557	44	18
B	37	6	13	2	**427**	0.93	DS17	427	427	427	1	3
B	38	5	13	2	**317**	1.25	DS17	317	317	317	1	<1
B	40	6	14	2	**469**	1.47	DS17	461	469	469	8	12
B	42	6	15	2	**405**	2.41	DS17	400	405	405	3	8
B	43	7	15	3	**443**	0.89	DS17	435	443	443	3	7
B	44	5	15	2	**489**	3.36	DS17	489	489	489	1	3
B	44	6	15	2	**386**	1.28	DS17	386	386	386	1	4
B	49	7	17	3	**464**	0.64	DS17	454	464	464	7	16
B	49	8	17	3	**661**	0.75	DS17	661	661	661	1	5
B	50	7	17	3	**578**	1.2	DS17	567	578	578	9	17
B	51	7	18	3	**427**	0.00	Bat14	421	427	427	3	11
B	55	7	19	3	**420**	3	DS17	416	420	420	4	16
B	56	7	19	3	**622**	1.89	DS17	582	622	622	138	437
B	56	9	19	3	**746**	0.93	DS17	709	746	746	505	1606
B	62	10	21	3	**685**	0.00	Bat14	685	685	685	1	21
B	63	9	22	4	**524**	0.38	DS17	522	524	524	5	32
B	65	9	22	3	**683**	0.58	DS17	666	683	683	30	252
B	66	10	23	4	**619**	1.12	DS17	610	619	619	11	72
B	67	9	23	3	**582**	1.02	DS17	582	582	582	1	25
B	77	10	26	4	**704**	2.36	DS17	679	704	704	91	1109

Table 3.11: Detailed results for the `GVRP-3` instances, subsets `A` and `B`.

Instance								Branch-and-Price results				
	n	k	N	m	BKS	*Gap to CluVRP*	*First found by*	LB root	LB tree	UB	#B&B nodes	Time T
P	15	8	6	4	**251**	0.79	DS17	251	251	251	1	<1
P	18	2	7	1	**170**	8.6	DS17	170	170	170	1	<1
P	19	2	7	1	**177**	11.5	DS17	177	177	177	1	<1
P	20	2	7	1	**179**	5.79	DS17	179	179	179	1	<1
P	21	2	8	1	**183**	9.41	DS17	183	183	183	1	<1
P	21	8	8	4	**365**	0.00	Bat14	365	365	365	1	<1
P	22	8	8	3	**270**	3.23	DS17	264	270	270	4	<1
P	39	5	14	2	**381**	3.79	DS17	380	381	381	8	8
P	44	5	15	2	**422**	4.09	DS17	422	422	422	1	2
P	49	7	17	3	**430**	3.8	DS17	430	430	430	1	4
P	49	8	17	3	**441**	4.13	DS17	441	441	441	1	3
P	49	10	17	4	**471**	4.07	DS17	470	471	471	3	5
P	50	10	17	4	**493**	8.19	DS17	493	493	493	1	2
P	54	7	19	3	**454**	1.73	B&P	453	454	454	4	21
P	54	8	19	3	**454**	3.61	B&P	454	454	454	2	9
P	54	10	19	4	**481**	3.8	DS17	481	481	481	2	4
P	54	15	19	6	**572**	3.87	DS17	570	572	572	9	9
P	59	10	20	4	**534**	3.26	B&P	530	534	534	21	61
P	59	15	20	5	**591**	3.27	DS17	585	591	591	14	39
P	64	10	22	4	**575**	7.11	B&P	575	575	575	1	8
P	69	10	24	4	**602**	6.38	DS17	602	602	602	1	30
P	75	4	26	2	**556**	4.3	B&P	554	556	556	20	382
P	75	5	26	2	**556**	4.3	DS17	556	556	556	1	71
P	100	4	34	2	**649**	4.42	DS17	648	649	649	6	1899
G	261	25	88	9	3196	-	DS17	-	-	-	0	3600
C	100	10	34	4	**598**	1.48	DS17	592	598	598	49	1707
C	120	7	41	3	681	-	DS17	-	-	-	0	3600
C	150	12	51	4	**756**	5.97	B&P†	-	-	-	0	3600
C	199	16	67	6	874	-	DS17	-	-	-	0	3600

Table 3.12: Detailed results for the `GVRP-3` instances, subsets `P` and `GC`.

3.C Detailed Results for the `Golden-Bat` Instances

Detailed results for the `Golden-Bat` instances are given in Tables 3.13 to 3.16, analogous to Section 3.B (without the number of vehicles k in the original CVRP instance).

Instance							Branch-and-Price results				
	n	N	m	BKS	*Gap to CluVRP*	*First found by*	LB root	LB tree	UB	#B&B nodes	Time T
Golden1	240	17	4	**4640**	3.95	B&P	4640	4640	4640	1	304
Golden1	240	18	4	4645	-	B&P	4610	4614	4645	10	3600
Golden1	240	19	4	4650	-	B&P	4621	4623	4650	6	3600
Golden1	240	21	4	4650	-	B&P†	4621	4621	-	4	3600
Golden1	240	22	4	4677	-	B&P†	4625	4625	-	3	3600
Golden1	240	25	4	4734	-	DS17	4609	4609	-	1	3600
Golden1	240	27	4	4708	-	B&P†	4620	4620	-	2	3600
Golden1	240	31	4	4766	-	DS17	4632	4632	-	1	3600
Golden1	240	35	4	4720	-	DS17	-	-	-	0	3600
Golden1	240	41	4	4710	-	DS17	-	-	-	0	3600
Golden1	240	49	4	4670	-	DS17	-	-	-	0	3600
Golden2	320	22	4	7434	-	B&P†	-	-	-	0	3600
Golden2	320	23	4	**7369**	4.21	B&P	7369	7369	7369	1	2836
Golden2	320	25	4	7369	-	B&P†	-	-	-	0	3600
Golden2	320	27	4	7480	-	DS17	-	-	-	0	3600
Golden2	320	30	4	7485	-	DS17	-	-	-	0	3600
Golden2	320	33	4	7471	-	DS17	-	-	-	0	3600
Golden2	320	36	4	7447	-	DS17	-	-	-	0	3600
Golden2	320	41	4	7450	-	DS17	-	-	-	0	3600
Golden2	320	46	4	7497	-	DS17	-	-	-	0	3600
Golden2	320	54	4	7487	-	DS17	-	-	-	0	3600
Golden2	320	65	4	7477	-	DS17	-	-	-	0	3600
Golden3	400	27	4	10389	-	DS17	-	-	-	0	3600
Golden3	400	29	4	10232	-	DS17	-	-	-	0	3600
Golden3	400	31	4	10273	-	DS17	-	-	-	0	3600
Golden3	400	34	4	10292	-	DS17	-	-	-	0	3600
Golden3	400	37	4	10255	-	DS17	-	-	-	0	3600
Golden3	400	41	4	10275	-	DS17	-	-	-	0	3600
Golden3	400	45	4	10252	-	DS17	-	-	-	0	3600
Golden3	400	51	4	10292	-	DS17	-	-	-	0	3600
Golden3	400	58	4	10350	-	DS17	-	-	-	0	3600
Golden3	400	67	4	10289	-	DS17	-	-	-	0	3600
Golden3	400	81	4	10311	-	DS17	-	-	-	0	3600
Golden4	480	33	4	13119	-	DS17	-	-	-	0	3600
Golden4	480	35	4	13205	-	DS17	-	-	-	0	3600
Golden4	480	37	4	13092	-	DS17	-	-	-	0	3600
Golden4	480	41	4	13011	-	DS17	-	-	-	0	3600
Golden4	480	44	4	13115	-	DS17	-	-	-	0	3600
Golden4	480	49	4	13133	-	DS17	-	-	-	0	3600
Golden4	480	54	4	13124	-	DS17	-	-	-	0	3600
Golden4	480	61	4	13190	-	DS17	-	-	-	0	3600
Golden4	480	69	4	13294	-	DS17	-	-	-	0	3600
Golden4	480	81	4	13235	-	DS17	-	-	-	0	3600
Golden4	480	97	4	13350	-	DS17	-	-	-	0	3600
Golden5	200	14	4	**6970**	8.55	B&P	6970	6970	6970	1	212
Golden5	200	15	3	**6742**	9.19	B&P	6742	6742	6742	1	280
Golden5	200	16	3	**6742**	10	B&P	6742	6742	6742	1	142
Golden5	200	17	3	**6862**	7.69	B&P	6862	6862	6862	1	380
Golden5	200	19	4	**6874**	9.27	B&P	6874	6874	6874	1	180
Golden5	200	21	4	**6816**	10.27	B&P	6816	6816	6816	1	666
Golden5	200	23	4	**6750**	11.68	B&P	6750	6750	6750	1	260
Golden5	200	26	4	**6704**	11.32	B&P	6704	6704	6704	1	647
Golden5	200	29	4	**6704**	9.53	B&P	6704	6704	6704	1	779
Golden5	200	34	4	**6684**	10.03	B&P*	-	-	-	0	3600
Golden5	200	41	4	**6557**	9.45	B&P	6557	6557	6557	1	2010

Table 3.13: Detailed results for the Golden-Bat instances 1-5.

Instance							Branch-and-Price results				
	n	N	m	BKS	*Gap to CluVRP*	*First found by*	LB root	LB tree	UB	#B&B nodes	Time T
Golden6	280	19	3	**8115**	5.9	B&P	8115	8115	8115	1	1110
Golden6	280	21	3	**8119**	5.9	B&P	8119	8119	8119	1	901
Golden6	280	22	3	**8107**	6.23	B&P	8107	8107	8107	1	1053
Golden6	280	24	4	**8316**	6.07	B&P	8316	8316	8316	1	2491
Golden6	280	26	4	**8249**	7.42	B&P	8249	8249	8249	1	2568
Golden6	280	29	4	8395	-	DS17	-	-	-	0	3600
Golden6	280	32	4	8290	-	DS17	-	-	-	0	3600
Golden6	280	36	4	8383	-	DS17	-	-	-	0	3600
Golden6	280	41	4	8405	-	DS17	-	-	-	0	3600
Golden6	280	47	4	8349	-	DS17	-	-	-	0	3600
Golden6	280	57	4	8461	-	DS17	-	-	-	0	3600
Golden7	360	25	3	**9318**	5.92	B&P†	-	-	-	0	3600
Golden7	360	26	3	**9295**	6	B&P†	-	-	-	0	3600
Golden7	360	28	3	9581	-	DS17	-	-	-	0	3600
Golden7	360	31	4	**9418**	6.02	B&P†	-	-	-	0	3600
Golden7	360	33	4	9685	-	DS17	-	-	-	0	3600
Golden7	360	37	4	**9395**	7.26	B&P†	-	-	-	0	3600
Golden7	360	41	4	9664	-	DS17	-	-	-	0	3600
Golden7	360	46	4	9642	-	DS17	-	-	-	0	3600
Golden7	360	52	4	9694	-	DS17	-	-	-	0	3600
Golden7	360	61	4	9713	-	DS17	-	-	-	0	3600
Golden7	360	73	4	9602	-	DS17	-	-	-	0	3600
Golden8	440	30	4	10651	-	DS17	-	-	-	0	3600
Golden8	440	32	4	10640	-	DS17	-	-	-	0	3600
Golden8	440	34	4	10682	-	DS17	-	-	-	0	3600
Golden8	440	37	4	10660	-	DS17	-	-	-	0	3600
Golden8	440	41	4	10692	-	DS17	-	-	-	0	3600
Golden8	440	45	4	10667	-	DS17	-	-	-	0	3600
Golden8	440	49	4	10732	-	DS17	-	-	-	0	3600
Golden8	440	56	4	10726	-	DS17	-	-	-	0	3600
Golden8	440	63	4	10747	-	DS17	-	-	-	0	3600
Golden8	440	74	4	10755	-	DS17	-	-	-	0	3600
Golden8	440	89	4	10714	-	DS17	-	-	-	0	3600
Golden9	255	18	4	**281**	6.33	B&P	280	281	281	19	1287
Golden9	255	19	4	**279**	6.69	B&P	279	279	279	2	209
Golden9	255	20	4	**276**	6.76	B&P	276	276	276	1	112
Golden9	255	22	4	**276**	4.83	B&P	276	276	276	1	217
Golden9	255	24	4	**276**	4.83	B&P	276	276	276	1	175
Golden9	255	26	4	**273**	5.21	B&P	273	273	273	3	465
Golden9	255	29	4	**273**	6.51	B&P	273	273	273	1	985
Golden9	255	32	4	**273**	8.08	B&P	273	273	273	1	1650
Golden9	255	37	4	**273**	7.14	B&P*	273	273	-	7	3600
Golden9	255	43	4	281	-	DS17	270	270	-	3	3600
Golden9	255	52	4	279	-	DS17	-	-	-	0	3600
Golden10	323	22	4	**346**	5.72	B&P	346	346	346	2	923
Golden10	323	24	4	**346**	4.16	B&P	346	346	346	3	1014
Golden10	323	25	4	**346**	3.62	B&P	346	346	346	6	1114
Golden10	323	27	4	**346**	4.16	B&P	346	346	346	4	1360
Golden10	323	30	4	**347**	5.45	B&P	347	347	347	2	1848
Golden10	323	33	4	**344**	7.77	B&P	344	344	344	1	2725
Golden10	323	36	4	**344**	10.65	B&P†	-	-	-	0	3600
Golden10	323	41	4	363	-	DS17	-	-	-	0	3600
Golden10	323	47	4	360	-	DS17	-	-	-	0	3600
Golden10	323	54	4	357	-	DS17	-	-	-	0	3600
Golden10	323	65	4	354	-	DS17	-	-	-	0	3600

Table 3.14: Detailed results for the Golden-Bat instances 6-10.

Instance							Branch-and-Price results				
	n	N	m	BKS	*Gap to CluVRP*	*First found by*	LB root	LB tree	UB	#B&B nodes	Time T
Golden11	399	27	5	**434**	5.03	B&P†	434	434	-	5	3600
Golden11	399	29	5	**434**	4.62	B&P*	434	434	-	3	3600
Golden11	399	31	5	**433**	4.84	B&P	433	433	433	1	2661
Golden11	399	34	5	**427**	6.15	B&P†	-	-	-	0	3600
Golden11	399	37	5	444	-	DS17	-	-	-	0	3600
Golden11	399	40	5	444	-	DS17	-	-	-	0	3600
Golden11	399	45	5	442	-	DS17	-	-	-	0	3600
Golden11	399	50	5	442	-	DS17	-	-	-	0	3600
Golden11	399	58	5	445	-	DS17	-	-	-	0	3600
Golden11	399	67	5	446	-	DS17	-	-	-	0	3600
Golden11	399	80	5	446	-	DS17	-	-	-	0	3600
Golden12	483	33	5	529	-	DS17	-	-	-	0	3600
Golden12	483	35	5	531	-	DS17	-	-	-	0	3600
Golden12	483	38	5	531	-	DS17	-	-	-	0	3600
Golden12	483	41	5	532	-	DS17	-	-	-	0	3600
Golden12	483	44	5	530	-	DS17	-	-	-	0	3600
Golden12	483	49	5	533	-	Bat14	-	-	-	0	3600
Golden12	483	54	5	535	-	Bat14	-	-	-	0	3600
Golden12	483	61	5	535	-	Vidal15	-	-	-	0	3600
Golden12	483	70	5	533	-	Vidal15	-	-	-	0	3600
Golden12	483	81	5	535	-	Vidal15	-	-	-	0	3600
Golden12	483	97	5	544	-	Vidal15	-	-	-	0	3600
Golden13	252	17	4	**530**	3.99	B&P	530	530	530	1	116
Golden13	252	19	4	**521**	5.1	B&P	521	521	521	1	189
Golden13	252	20	4	**521**	4.93	B&P	521	521	521	1	192
Golden13	252	22	4	**523**	4.56	B&P	523	523	523	1	203
Golden13	252	23	4	**523**	4.56	B&P	523	523	523	1	215
Golden13	252	26	4	**523**	3.51	B&P	523	523	523	1	118
Golden13	252	29	4	**522**	3.33	B&P	522	522	522	5	1483
Golden13	252	32	4	**521**	4.05	B&P	521	521	521	1	286
Golden13	252	37	4	**521**	4.4	B&P	521	521	521	3	2305
Golden13	252	43	4	**521**	5.79	B&P†	-	-	-	0	3600
Golden13	252	51	4	532	-	DS17	-	-	-	0	3600
Golden14	320	22	4	**665**	3.9	B&P	665	665	665	3	1814
Golden14	320	23	4	**662**	3.78	B&P	662	662	662	2	752
Golden14	320	25	4	**660**	2.65	B&P	660	660	660	1	637
Golden14	320	27	4	**660**	2.37	B&P	660	660	660	11	3067
Golden14	320	30	4	**660**	2.65	B&P†	660	660	-	5	3600
Golden14	320	33	4	672	-	DS17	660	660	-	3	3600
Golden14	320	36	4	668	-	DS17	-	-	-	0	3600
Golden14	320	41	4	**658**	4.64	B&P†	-	-	-	0	3600
Golden14	320	46	4	676	-	DS17	-	-	-	0	3600
Golden14	320	54	4	674	-	DS17	-	-	-	0	3600
Golden14	320	65	4	679	-	DS17	-	-	-	0	3600
Golden15	396	27	4	825	-	DS17	-	-	-	0	3600
Golden15	396	29	4	825	-	DS17	815	815	-	1	3600
Golden15	396	31	4	**813**	2.87	B&P	813	813	813	2	3176
Golden15	396	34	4	826	-	DS17	-	-	-	0	3600
Golden15	396	37	4	826	-	DS17	-	-	-	0	3600
Golden15	396	40	4	830	-	DS17	-	-	-	0	3600
Golden15	396	45	5	834	-	DS17	-	-	-	0	3600
Golden15	396	50	5	839	-	DS17	-	-	-	0	3600
Golden15	396	57	5	838	-	DS17	-	-	-	0	3600
Golden15	396	67	5	840	-	DS17	-	-	-	0	3600
Golden15	396	80	5	843	-	DS17	-	-	-	0	3600

Table 3.15: Detailed results for the Golden-Bat instances 11-15.

Instance							Branch-and-Price results				
	n	N	m	BKS	*Gap to CluVRP*	*First found by*	*LB* root	*LB* tree	*UB*	#B&B nodes	Time T
Golden16	480	33	5	1013	-	DS17	-	-	-	0	3600
Golden16	480	35	5	1011	-	DS17	-	-	-	0	3600
Golden16	480	37	5	1007	-	DS17	-	-	-	0	3600
Golden16	480	41	5	1013	-	DS17	-	-	-	0	3600
Golden16	480	44	5	1018	-	DS17	-	-	-	0	3600
Golden16	480	49	5	1014	-	DS17	-	-	-	0	3600
Golden16	480	54	5	1012	-	DS17	-	-	-	0	3600
Golden16	480	61	5	1012	-	DS17	-	-	-	0	3600
Golden16	480	69	5	1012	-	Bat14	-	-	-	0	3600
Golden16	480	81	5	1017	-	DS17	-	-	-	0	3600
Golden16	480	97	5	1018	-	Bat14	-	-	-	0	3600
Golden17	240	17	3	**386**	7.66	B&P	386	386	386	1	132
Golden17	240	18	3	**385**	8.11	B&P	385	385	385	1	290
Golden17	240	19	3	**385**	8.77	B&P	385	385	385	1	220
Golden17	240	21	3	**385**	9.41	B&P	385	385	385	2	457
Golden17	240	22	3	**385**	9.2	B&P	385	385	385	1	372
Golden17	240	25	3	**382**	8.61	B&P	382	382	382	1	487
Golden17	240	27	3	**382**	7.73	B&P	382	382	382	3	1039
Golden17	240	31	4	**390**	7.36	B&P	390	390	390	1	661
Golden17	240	35	4	396	-	B&P†	389	389	-	2	3600
Golden17	240	41	4	**388**	5.83	B&P†	-	-	-	0	3600
Golden17	240	49	4	396	-	DS17	-	-	-	0	3600
Golden18	300	21	4	**558**	5.74	B&P	558	558	558	1	694
Golden18	300	22	4	**558**	6.06	B&P	558	558	558	1	781
Golden18	300	24	4	**558**	5.74	B&P	558	558	558	1	831
Golden18	300	26	4	**562**	4.75	B&P	562	562	562	1	974
Golden18	300	28	4	**558**	3.29	B&P*	-	-	-	0	3600
Golden18	300	31	4	**554**	4.15	B&P	554	554	554	1	2450
Golden18	300	34	4	**554**	4.81	B&P	554	554	554	1	1992
Golden18	300	38	4	**555**	5.29	B&P†	-	-	-	0	3600
Golden18	300	43	4	573	-	DS17	-	-	-	0	3600
Golden18	300	51	4	575	-	DS17	-	-	-	0	3600
Golden18	300	61	4	574	-	DS17	-	-	-	0	3600
Golden19	360	25	10	**886**	4.22	B&P	886	886	886	1	538
Golden19	360	26	10	**888**	3.9	B&P	888	888	888	1	1208
Golden19	360	28	4	**741**	8.29	B&P	741	741	741	1	1479
Golden19	360	31	4	**735**	9.37	B&P*	-	-	-	0	3600
Golden19	360	33	4	**727**	8.78	B&P	727	727	727	1	2719
Golden19	360	37	5	**732**	8.39	B&P	732	732	732	1	2612
Golden19	360	41	5	**730**	7.48	B&P†	-	-	-	0	3600
Golden19	360	46	5	752	-	DS17	-	-	-	0	3600
Golden19	360	52	5	**730**	8.75	B&P†	-	-	-	0	3600
Golden19	360	61	5	763	-	DS17	-	-	-	0	3600
Golden19	360	73	5	763	-	DS17	-	-	-	0	3600
Golden20	420	29	11	**1170**	4.1	B&P	1170	1170	1170	1	1099
Golden20	420	31	12	**1183**	3.98	B&P	1183	1183	1183	1	1080
Golden20	420	33	12	**1175**	2.73	B&P	1175	1175	1175	1	2381
Golden20	420	36	5	1033	-	DS17	-	-	-	0	3600
Golden20	420	39	5	1025	-	DS17	-	-	-	0	3600
Golden20	420	43	5	1017	-	DS17	-	-	-	0	3600
Golden20	420	47	5	1023	-	DS17	-	-	-	0	3600
Golden20	420	53	5	1022	-	DS17	-	-	-	0	3600
Golden20	420	61	5	1021	-	DS17	-	-	-	0	3600
Golden20	420	71	5	1025	-	DS17	-	-	-	0	3600
Golden20	420	85	5	1018	-	DS17	-	-	-	0	3600

Table 3.16: Detailed results for the Golden-Bat instances 16-20.

Chapter 4

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

Timo Hintsch

Abstract

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

4.1 Introduction

The *soft-clustered vehicle-routing problem* (SoftCluVRP) is a variant of the well-known *capacitated vehicle-routing problem* (CVRP, Toth and Vigo, 2014) and has been introduced by Defryn and Sörensen (2017). It can be described as follows. The customers are grouped into disjoint clusters and all customers of a cluster must be served by the same vehicle (*soft-cluster constraints*). Visits to customers of the same cluster can be interrupted by visits to customers of other clusters. This is a relaxation of the *clustered vehicle-routing problem* (CluVRP, Sevaux and Sörensen, 2008) in which interruption is not allowed, but all customers of a cluster must be served contiguously (*hard-cluster constraints*). In Chapter 3 we have shown that this relaxation can decrease the costs of optimal solutions by 6.21 % on average for medium-sized instances, but finding optimal solutions is very difficult.

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

The CluVRP is addressed by exact approaches (Pop *et al.*, 2012; Battarra *et al.*, 2014a) and by several metaheuristics (Barthélemy *et al.*, 2010; Expósito Izquierdo *et al.*, 2013; Vidal *et al.*, 2015; Expósito-Izquierdo *et al.*, 2016; Defryn and Sörensen, 2017; Pop *et al.*, 2018; this thesis, Chapter 2). To the best of our knowledge, only two approaches consider the SoftCluVRP. In Chapter 3 we presented an exact branch-and-price algorithm, which provides optimal solutions for instances with up to 420 customers and up to 52 clusters. Defryn and Sörensen (2017) suggested a two-level metaheuristic that originally was developed for the CluVRP. In this case, the low-level routing problem only considers the routing of customers inside a cluster (intra-cluster routing) and the high-level routing problem alters the position of clusters inside a route or moves clusters to another route (inter-cluster routing). This approach was adapted to the SoftCluVRP by allowing for customers to be moved to any position inside the current route at the lower level. Hence, the low-level routing considers intra-route moves of customers and the high-level routing considers inter-route moves of complete clusters. Both levels are solved by variable neighborhood search (VNS, Mladenović and Hansen, 1997).

The main contribution of this chapter is the design and computational analysis

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

Large neighborhood search (LNS, Shaw, 1998; Ropke and Pisinger, 2006b) has been shown to solve a wide range of routing problems successfully (see the survey by Pisinger and Ropke, 2010). Our approach combines the usage of multiple destroy and repair operators with two variable neighborhood descents (VNDs, Hansen and Mladenović, 2001) for post-optimization. The first VND allows for relocating and swapping complete clusters between routes, while the second VND improves single routes by classical neighborhoods (2-opt, Or-opt, and double-bridge, see, e.g., Funke *et al.*, 2005) for the asymmetric traveling salesman problem (ATSP) as well as the Balas-Simonetti neighborhood (Balas, 1999; Balas and Simonetti, 2001). Although it is of exponential size, the Balas-Simonetti neighborhood can be searched in polynomial time.

The general design of our approach is adapted from the LMNS for the CluVRP presented in Chapter 2. However, important components of the approach in Chapter 2 are based on the exploitation of the hard-cluster constraints, for example the preprocessing of intra-cluster routes, a meta-representation with meta-nodes for the clusters, and a generalization of the Balas-Simonetti neighborhood. Since we consider soft-cluster constraints in this chapter, major modifications are required for the destroy and repair operators as well as the post-optimization, resulting in a clearly differing algorithm (see Section 4.2).

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

The task is to find m feasible routes visiting each customer exactly once and minimizing the total routing costs. According to the literature (e.g. Battarra *et al.*, 2014a) and the previous chapters of this thesis, each vehicle has to serve at least one cluster. Hence, a route r is feasible if

(i) it starts and ends at the depot node 0 and serves at least one cluster V_h, $h \in H$,

(ii) it visits each customer $i \in V_h$ exactly once if any customer $j \in V_h$ is visited by r, and
(iii) the demand of the visited clusters respects the vehicle capacity Q.

The remainder of this chapter is structured as follows. In Section 4.2, the overall LMNS algorithm and all its components are described in detail. Comprehensive computational studies are summarized in Section 4.3. We analyze the effects of the destroy and repair operators as well as both post-optimization components. Moreover, we compare the results of the LMNS to the results generated by Defryn and Sörensen (2017). Final conclusions are drawn in Section 4.4.

4.2 LMNS for the SoftCluVRP

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

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

As an extension, Ropke and Pisinger (2006a) introduced the *adaptive* LNS (ALNS) for the pickup and delivery problem with time windows. In each iteration, the destroy and the repair operator are selected randomly out of a set of multiple destroy and repair operators depending on a given weight per operator. The weights are updated according to the success of the respective operators in former iterations. LMNS (Pisinger and Ropke, 2007) also uses different destroy and repair operators, but in contrast to ALNS their given weights remain unchanged. Our approach is an adaptation of the LMNS for the CluVRP developed in Chapter 2. We adopt the record-to-record acceptance criterion and the idea of post-optimizing the repaired solution (which was first suggested by Ropke, 2009). However, due to the soft-cluster constraints, customers of clusters that are served by the same route can be visited in arbitrary order and a meta-representation of routes on a cluster level is not applicable. The meta-representation was an essential property of the LMNS in Chapter 2. Hence, we have to implement four major modifications:

(i) We cannot exploit the pre-computation of intra-cluster routes. Instead, we calculate feasible routes that include the depot and serve one cluster or a pair of clusters. These routes are used during the construction phase and possibly during the repair and the post-optimization phase.
(ii) The destroy and repair operators have to be tailored to the SoftCluVRP.
(iii) Similarly, new variants of the cluster neighborhoods are presented and combined in a VND for post-optimization (called `Clu-VND` in the following).
(iv) The generalized version of the Balas-Simonetti neighborhood, used during and after the VND, cannot be employed. Instead, we extend the post-optimization phase by a second VND which combines classical neighborhoods with the basic Balas-Simonetti neighborhood. This VND searches for intra-route improvements and is called `ATSP-VND` in the following.

In the following, we describe all our LMNS components. Section 4.2.1 presents improvement strategies for single routes, including the `ATSP-VND`, and Section 4.2.2 combines two neighborhoods that exchange clusters between different routes to another VND, called `Clu-VND`. In Section 4.2.3, we introduce our destroy and repair operators. Subsequently, the overall LMNS is summarized in Section 4.2.4.

4.2.1 ATSP Heuristics

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

In this section, we present a simple VND for the ATSP, which is used in our LMNS as a post-optimization component. It is based on three classical edge-exchange neighborhoods (2-opt, Or-opt, and double-bridge) and the Balas-Simonetti neighborhood. Furthermore, we embed the VND in an iterated local search (ILS, Johnson *et al.*, 2007), which results in a combined ILS/VND similar to the algorithm presented by Irnich (2008). This procedure is used during the construction phase (see Section 4.2.4). Before explaining the `ATSP-VND` and the `Combined-ILS/VND`, we give a short description on the Balas-Simonetti neighborhood.

Balas-Simonetti neighborhood The Balas-Simonetti neighborhood $\mathcal{N}_k^{BS}$ was introduced by Balas (1999) and is defined for a given integer parameter $k \geq 2$. Let $r = (r_0 = 0, r_1, \ldots, r_{n_r}, r_{n_r+1} = 0)$ be a feasible route. Then, if r_i precedes r_j in r by at least k positions, node r_i must also precede node r_j in a neighbor

route $r' \in \mathcal{N}_k^{BS}(r)$. Hence, the Balas-Simonetti neighborhood $\mathcal{N}_k^{BS}(r)$ comprises all routes r' in which (i) $r'_0 = r_0$ and $r'_{n_r+1} = r_{n_r+1}$, and (ii) for all $i, j \in \{1, \ldots, n_r\}$ with $i + k \leq j$, node r_i precedes node r_j also in r'. A layered auxiliary network is constructed to find the best neighbor route, where each *network node* refers to a combination of a node r_i of the current route r and a (possibly new) position i' in route r', for which $i - k < i' < i + k$ holds. Each source-sink path in the auxiliary network represents a feasible neighbor route $r' \in \mathcal{N}_k^{BS}(r)$.

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

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

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

$n_r \leq 2$: There is nothing to do. The resulting route is a *pendulum tour* including the depot and the only customer node (or two customer nodes); note that we consider symmetric instances.

$3 \leq n_r \leq n^{small}$: We construct an arbitrary starting route r and search for the best neighbor route $r' \in \mathcal{N}_{n_r}^{BS}(r)$ only once. Since we set $k = n_r$ for the Balas-Simonetti neighborhood, the resulting route is already optimal.

$n_r > n^{small}$: The actual combination of ILS and VND similar to the procedure by Irnich (2008) is applied: First, a starting route is constructed by the nearest neighbor heuristic. Second, we iteratively call `ATSP-VND`(k) and permute the derived local optimum by two random double-bridge moves. The result of the permutation is used as the new starting solution for the next iteration. Overall, It_{ILS} iterations are executed, while keeping track of the best solution found.

4.2.2 Cluster Neighborhoods and VND

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

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

The reinsertion of cluster V_h into a given route r is feasible if d_h, the demand of cluster V_h, does not exceed the residual capacity of r. Only feasible insertions are considered. To reinsert the cluster V_h, all customers $i \in V_h$ are sorted randomly. Then, they are inserted one after another by the Procedure `Best Insert`. A single customer is inserted into the current route by minimizing the insertion cost. Note that the computational effort is bounded by $\mathcal{O}(n_{\max} n)$, where $n_{\max}$ is the size of the largest cluster.

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

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

Swap Neighborhood The neighborhood $\mathcal{N}^{\text{swap}}$ contains all SoftCluVRP solutions that result from the swapping of two clusters from two different routes. A swap of cluster V_g, currently visited by route r, and cluster V_h, currently visited by route s, is performed as follows: First, we remove all nodes $i \in V_g \cup V_h$ from their current route. Second, we perform `Best Insert`(V_g^{ran}, s) and `Best`

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

Input: Randomly sorted customers V_h^{ran} of cluster V_h,
a route r.

Output: Insertion costs and new route r including all customers in V_h

```
1  for i ∈ V_h^ran do
2      c_min := ∞
3      pos := −1
4      for j := 0, ..., n_r do
5          cost := c_{r_j,i} + c_{i,r_{j+1}} − c_{r_j,r_{j+1}}
6          if cost < c_min then
7              c_min = cost
8              pos = j + 1
9      r' := (r_0, ..., r_{pos−1}, i, r_pos, ..., r_{n_r}, r_{n_r+1})
10     r := r'
```

`Insert`(V_h^{ran}, r). The size of $\mathcal{N}^{\text{swap}}$ grows quadratically with the number of clusters N and the computational effort is limited to $\mathcal{O}(2n_{\max}nN^2)$.

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

4.2.3 LNS Operators

Here, we describe the different destroy and repair operators that are employed in our LMNS.

Destroy Operators The destroy operators always remove complete clusters, which means that each customer $i \in V_h$ is removed if cluster V_h is removed. Removing a cluster is performed as described in the previous section. The percentage of clusters to be removed is defined by a parameter τ and we use four different destroy operators, similar to the operators applied in Chapter 2:

1. *Random destroy* removes τN clusters at random (Ropke and Pisinger, 2006a). (Note that τN is always rounded to the next integer. Here, we omit writing $\lfloor \tau N + 0.5 \rfloor$ for simplicity.)

2. *Related destroy* was introduced by Shaw (1998) and we adapt it to the presence of clusters: First, one cluster V_h is removed at random. Then, $\tau N - 1$

clusters closest to V_h are removed, too. The distance between two clusters V_g and V_h is defined as $\min_{(i,j)\in V_g \times V_h} c_{ij}$.

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

4. *Route destroy* removes one entire route at random (Gschwind and Drexl, 2019). Note that the parameter τ is not used by this operator.

Repair Operators As the destroy operators do, the repair operators reinsert complete clusters. All operators use the same procedure to insert a given cluster V_h: If the destroy operator has reduced the number of routes and not every vehicle serves at least one cluster in the current solution, the given cluster is used to start a new route. Otherwise, for each route where V_h could be inserted w.r.t. the capacity, we evaluate the insertion costs for cluster V_h by the Procedure `Best Insert` as described in Section 4.2.2. Afterwards, the route with smallest insertion cost is chosen and cluster V_h is inserted as determined before. If cluster V_h cannot be inserted into any route because of the capacity constraint, the repair operator is stopped and the current solution remains infeasible. The operators only differ in the order the clusters are inserted:

1. *Random repair* reinserts all removed clusters in random order (Schrimpf *et al.*, 2000).

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

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

4.2.4 Overall LMNS Algorithm

Our overall LMNS approach combines all components described in Sections 4.2.1–4.2.3. Next, we describe the pseudo-code that is given in Algorithm 6:

In Step 1, we employ a *savings algorithm* (Clarke and Wright, 1964), tailored to the SoftCluVRP, to construct a starting solution x. In contrast to the classical savings algorithm, a *pendulum tour* is defined as a route visiting all customers of one cluster, instead of visiting only one customer. For each cluster $V_h, h \in H$, we calculate a route, starting and ending in the depot, and visiting all customers $i \in V_h$ by applying the `Combined-ILS/VND` (Section 4.2.1) with the given input parameters. Moreover, the same is done for each pair of clusters (V_g, V_h) with $(g, h) \in H \times H$. Such a route visits all customers $i \in V_g \cup V_h$ of both clusters. The costs of the resulting routes are defined as $\hat{c}_h$ and $\hat{c}_{g,h}$, respectively, and savings values are calculated for each pair (g, h) as $sav_{g,h} = \hat{c}_g + \hat{c}_h - \hat{c}_{g,h}$.

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

Afterwards, the main loop (Steps 2–14) runs for It_{LMNS} iterations. Note that infeasible solutions can occur after the destroy/repair phase, but we only accept feasible solutions (Step 3). Furthermore, given a parameter ϵ_{post}, we only accept promising solutions that fulfill the record-to-record criterion $c(x) < (1 + \epsilon_{\text{post}})\, c(x^{best})$ for post-optimization, where $c(x)$ is defined as the cost of solution x (Step 4). The post-optimization is performed in Steps 5 and 6 with the `Clu-VND` from Section 4.2.2 followed by the `ATSP-VND` from Section 4.2.1. The latter is called for each route of the current solution x and with $k = k_{\text{post}}$.

In Steps 7–10, we possibly update the best solution found and/or the accepted solution depending on the second acceptance criterion. Again, we use a record-to-record acceptance criterion and the current solution is accepted if $c(x) < (1 + \epsilon_{\text{LMNS}})\, c(x^{best})$ is fulfilled for a given parameter ϵ_{LMNS}. Note that we always set $\epsilon_{\text{post}} \geq \epsilon_{\text{LMNS}}$. Steps 11–13 randomly choose the percentage τ of clusters to be removed as well as one destroy and one repair operator out of the seven operators presented in Section 4.2.3. Afterwards, the chosen operators are applied in Step 14,

Algorithm 6: LMNS algorithm for the SoftCluVRP

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

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

resulting in the new solution for the next iteration.

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

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

4.3 Computational Results

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

In Section 4.3.1, we introduce the considered benchmark instances and in Section 4.3.2, the parameters of our LMNS are tuned. Afterwards, the calibrated LMNS is analyzed and compared to the two-level VNS by Defryn and Sörensen (2017) on different instance sets (Sections 4.3.3 and 4.3.4). Results for large-sized instances that were not considered for the SoftCluVRP in the literature before are presented in Section 4.3.5.

4.3.1 SoftCluVRP Benchmark Instances

We test our LMNS algorithm on three benchmark sets that were used in previous studies in the literature. All SoftCluVRP benchmark sets were derived from CVRP benchmarks by defining θ as the desired average number of customers per customer cluster and building $N = \lceil (n+1)/\theta \rceil$ customer clusters (for details, see Fischetti *et al.*, 1997; Bektaş *et al.*, 2011). The first benchmark set was proposed by Bektaş *et al.* (2011). They adapted the CVRP benchmarks called `A`, `B`, `P`, and `GC` by choosing $\theta \in \{2, 3\}$, resulting in the two subsets `GVRP-2` and `GVRP-3`. Overall, 158 small- and medium-sized instances (available online at `http://www.personal.soton.ac.uk/tb12v07/gvrp.html`) with 16 to 262 nodes and 6 to 131 clusters were generated. The second benchmark set `Golden` is based on the well-known CVRP instances by Golden *et al.* (1998) and was generated by choosing $\theta = \{5, \ldots, 15\}$ for each of the 20 original instances `Golden1` to `Golden20`. It was provided by Battarra *et al.* (2014a) and consists of 220 large-scale instances with 201 to 484 nodes and 14 to 97 clusters. The third set `Li` was generated by Vidal *et al.* (2015) using the CVRP instances of Li *et al.* (2005) and $\theta = 5$. It comprises 12 large-scale instances with 561 to 1201 nodes and 113 to 241 clusters. The

number of vehicles m is given for each instance and it is not allowed to use less vehicles.

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

4.3.2 Parameter Studies

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

Parameters for the Savings Algorithm

The only parameters that need to be set for the savings algorithm are those of the `Combined-ILS/VND`: $n^{small}, It_{\text{ILS}}, k_{\text{sav}}$. We simply adopt the parameter settings chosen in Chapter 2, which turned out to be a good tradeoff between solution quality and computational effort. In Chapter 2, the `Combined-ILS/VND` was used to compute the shortest Hamiltonian path for each pair of nodes inside a cluster, which played an important role for the overall algorithm. In this chapter, it is only used during the construction phase. Although the derived routes might be used during the overall algorithm (see Section 4.2.4), the results are not crucial for the LMNS of this chapter. In contrast to the approach of Chapter 2, they can be corrected by later steps.

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

Increasing the number of iterations It_{ILS} from 50 to 100 even leads to an inferior solution quality with *Gap Best* = 0.039 % and *Gap Avg.* = 0.138 % with the same computational effort (*Geo.* T = 17.0).

Parameters for the basic LMNS

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

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

Subsequently, we systematically test for the usefulness of each and every operator. We compare the chosen setup (called *All Operators*) to setups where one of the operators is disabled, but the ratio of the remaining operators is kept fixed. For example, if the *Worst destroy* is disabled, the weights for the destroy operators change to $(\psi^{random}, \psi^{related}, \psi^{route}) = (0.3, 0.3, 0.1)/0.7$. Again, we set $k_{\text{post}} = 3$ and It_{LMNS} = 10 000. The results are summarized in Table 4.1. *Avg.* T refers to the arithmetic mean of the average computation time over ten runs for all 158

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

Table 4.1: Comparison of LMNS using different destroy and repair operators and 158 `GVRP` benchmark instances.

instances and *Geo. T* gives the geometric mean.

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

Usefulness of the post-optimization

In our LMNS, the post-optimization of repaired solutions comprises two components: The `Clu-VND`, which relocates and swaps complete clusters, and the `ATSP-VND`, which improves single routes and consists of four neighborhoods (2-opt, Or-opt, double-bridge, Balas-Simonetti). In addition to the `GVRP` instances, we include the medium-sized `Golden` instances for the analysis of the two components. Altogether, we compare the following 15 post-optimization strategies:

`Full-`$k_{\texttt{post}}$: The full LMNS is applied as described in Section 4.2.4, including the `Clu-VND` and the `ATSP-VND`. For the `ATSP-VND`, we test 5 different settings with $k_{\text{post}} \in \{0, 3, 5, 7, 9\}$, where $k_{\text{post}} = 0$ means that the Balas-Simonetti neighborhood is switched off. Note that $k_{\text{post}} = 1$ is not reasonable because nodes could only be moved for *less* than $k_{\text{post}} = 1$ positions in the current route, which corresponds to not move them at all.

`BSLS-`$k_{\texttt{post}}$: Instead of using the complete `ATSP-VND`, we only search for the local optimum with respect to the Balas-Simonetti neighborhood for each route, using different $k_{\text{post}} \in \{3, 5, 7, 9\}$. Note that $k_{\text{post}} = 0$ would switch off the `ATSP-VND` completely,

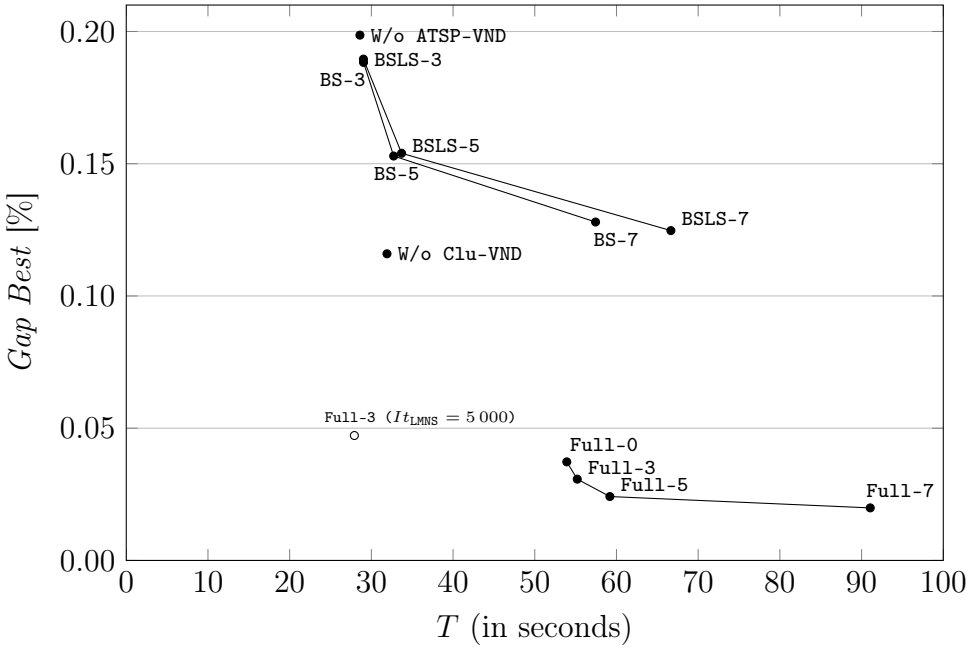

Figure 4.1: Comparison of different post-optimization strategies on 378 benchmarks (158 `GVRP` and 220 `Golden` instances).

which is the strategy `W/o ATSP-VND`.

`BS-`$k_{\texttt{post}}$: As `BSLS-`$k_{\texttt{post}}$, but instead of searching for the local optimum, the Balas-Simonetti neighborhood is applied only once for each route.

`W/o ATSP-VND`: The complete `ATSP-VND` is switched off. Only `Clu-VND` is used for post-optimization.

`W/o Clu-VND`: `Clu-VND` is switched off. Only the (full) `ATSP-VND` with $k_{\text{post}} = 3$ is used for post-optimization.

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

Figure 4.1 gives a comparison of the different post-optimization strategies when running the LMNS with 10 000 iterations. It reports the average computation time *Avg. T* and the average *Gap Best* for all the 378 `GVRP` and `Golden` instances. To compare for similar computation times, we additionally run the setting `Full-3` with a reduced number of iterations $It_{\text{LMNS}} = 5\,000$, indicated by the open dot ∘.

The results can be summarized as follows: First, it is superior to post-optimize solutions with the `ATSP-VND` including the three classical edge-exchange neighbor-

hoods. All settings where ATSP-VND is switched off or reduced to a local/single search with the Balas-Simonetti neighborhood are clearly outperformed w.r.t. *Gap Best*. Reducing the number of iterations, e.g., of setting Full-3, shows that this also holds when computation times are comparable. Second, a similar effect is observed for the Clu-VND. Comparing the strategies W/o Clu-VND and Full-3, the Clu-VND decreases the *Gap Best* from 0.116 % to 0.031 % but increases the *Avg. T* from 31.9 to 55.2 seconds. As before, reducing the number of iterations for Full-3 helps to show the usefulness of Clu-VND by producing a *Gap Best* of 0.047 % in 27.9 seconds average computation time. Third, we observe the expected tradeoff between solution quality and computation time for the k_{post} of the Balas-Simonetti neighborhood used in the ATSP-VND. Higher values for k_{post} help to find better solutions, while the computational effort only raises reasonably for $k_{\text{post}} \leq 5$. However, for $k_{\text{post}} > 5$, computation times increase drastically with only little improvement in the solution quality. We omit settings with $k_{\text{post}} = 9$ in Figure 4.1 due to very high running times. For example, Full-9 gives the smallest *Gap Best* of only 0.018 % but runs for 276.3 seconds on average.

Very similar results are observed by comparing *Gap Avg.* instead of *Gap Best*. Overall, both VND components used for post-optimization contribute to the quality of our LMNS. Furthermore, setting k_{post} to 3 or 5 yields the most favorable results and we report more details on both settings in the next sections. This result is very similar to observations from the literature (see, e.g., Gschwind and Drexl, 2019, and Chapter 2 of this thesis).

4.3.3 Results for the GVRP Instances

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

The results are summarized in Table 4.2, where 'DS (2017)' refers to the two-level VNS by Defryn and Sörensen (2017). Our two LMNS settings perform very similar on these instances. In total, the computation time is smaller for $\text{LMNS}^3_{10\,000}$, but it differs less than one second on average. Overall, $\text{LMNS}^3_{10\,000}$ ($\text{LMNS}^5_{10\,000}$) finds the BKS for all but one (two) instance(s), resulting in an average *Gap Best* of <0.01 % (<0.01 %). For the GC-2 instances, we obtain an average *Gap Best* of 0.01 % (0.16 %). Considering *Gap Avg.*, $\text{LMNS}^5_{10\,000}$ performs slightly better (0.08 % vs. 0.09 % overall).

Comparing with the two-level VNS, note that Defryn and Sörensen (2017) run their algorithm for 20 different random seeds, but did not consider the GVRP-2 instances. Furthermore, they reported computation times only by arithmetic means for subsets. For the GVRP-3 instances, they can find the BKS for 69 instances,

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

Table 4.2: Aggregated results for the 158 `GVRP` instances.

whereas both LMNS settings find each and every BKS, resulting in smaller or equal *Gap Best* values. The *Gap Avg.* values are smaller for the LMNS, too, except for the `B-3` instances (0.02 % vs. 0.00 %). Computation times are clearly smaller for the two-level VNS, but also reasonably small for both LMNS settings and all subsets `A-3`, `B-3`, `P-3` (at most 3.28 seconds on average). Moreover, reducing the number of iterations to 1 000 and applying additional tests, for example with $\text{LMNS}^3_{1\,000}$, leads to average computation times that are smaller for our LMNS for each of the subsets except `B-3` (0.27 vs. 0.06 seconds), and we can still find every BKS.

Furthermore, our LMNS finds every solution that is known to be optimal (for optimal solutions, see Chapter 3) and improves the BKS from the literature for 10 ($\text{LMNS}^3_{10\,000}$) and 11 ($\text{LMNS}^5_{10\,000}$) of the remaining 13 instances (where the exact approach was prematurely terminated after 3 600 seconds). One of the new BKS can even be proven to be an optimal solution, since the calculated cost equals the corresponding lower bound reported for this instance in Chapter 3. Detailed instance-by-instance results are given in Tables 4.5–4.8 in Section 4.A of the Appendix.

4.3.4 Results for the `Golden` Instances

Analogous to the previous section, we analyze our LMNS for the medium-sized `Golden` instances. Results are given in Table 4.3 (for $k_{\text{post}} = 3$) and Table 4.4 (for $k_{\text{post}} = 5$), where the instances are grouped by average cluster size $\theta \in \{5, \ldots, 15\}$. Instances are easier to solve for larger average cluster sizes θ, which implies a decreasing number of clusters N. This leads to strictly decreasing computation times for both LMNS settings. The gaps also tend to decrease with an increasing

θ, but this observation is ambiguous, in particular for *Gap Best*.

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

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

Table 4.3: Aggregated results for $k_{\text{post}} = 3$ and benchmark set `Golden`, sorted by the average number of nodes per cluster θ (220 instances divided into 11 groups of 20 instances each).

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

Furthermore, both our LMNS settings produce 130 new BKS for the `Golden`

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

Table 4.4: Aggregated results for $k_{\text{post}} = 5$ and benchmark set `Golden`, sorted by the average number of nodes per cluster θ (220 instances divided into 11 groups of 20 instances each).

instance set. Out of them, 7 solutions are optimal because they hit the corresponding lower bound generated by the branch-and-price algorithm in Chapter 3. In addition, 5 new solutions that were generated during the computational experiments are also proven to be optimal. Overall, our LMNS with setting $\text{LMNS}^{3}_{10\,000}$ ($\text{LMNS}^{5}_{10\,000}$) finds 81 (82) out of the 99 solutions for `Golden` instances that are now known to be optimal. Detailed results for each instance are given in Section 4.B of the Appendix (Tables 4.9–4.12).

Finally note that, compared to the BKS reported for the CluVRP in the literature, heuristic solutions generated with our LMNS for the SoftCluVRP (e.g. with setting $\text{LMNS}^{5}_{10\,000}$) reduce the costs by 6.19 % on average over all `Golden` instances. If we only consider instances that are solved exactly for both problem variants, the cost reduction is 6.10 % on average. See Chapter 3 for a more detailed comparison of hard- and soft-cluster constraints on exactly solved instances.

4.3.5 Results for the `Li` Instances

In a subsequent study, we run our LMNS with both settings, $\text{LMNS}^{3}_{10\,000}$ and $\text{LMNS}^{5}_{10\,000}$, on the 12 large-sized `Li` instances (see Table 4.13 in Section 4.C of the Appendix for detailed instance-by-instance results). These were not solved for the SoftCluVRP before. $\text{LMNS}^{3}_{10\,000}$ finds the better result for 7 instances, while $\text{LMNS}^{5}_{10\,000}$ finds better solutions for the remaining 5 instances. The resulting gaps

are *Gap Best* = 0.02 % (*Gap Avg.* = 0.36 %) within 658 seconds of average runtime for $\text{LMNS}^3_{10\,000}$ and *Gap Best* = 0.03 % (*Gap Avg.* = 0.31 %) within 680 seconds for $\text{LMNS}^5_{10\,000}$. Hence, $\text{LMNS}^3_{10\,000}$ performs slightly better on these instances, but note that they were all generated by choosing $\theta = 5$ and $\text{LMNS}^3_{10\,000}$ also performed better on this group of the `Golden` instances.

Compared to the BKS for the CluVRP (see Vidal *et al.*, 2015, and Chapter 2 of this thesis), costs for the `Li` instances are reduced by up to 7.38 % (4.75 % on average) due to the soft-cluster relaxation.

4.4 Conclusions

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

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

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Gschwind, T. and Drexl, M. (2019). Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, **53**(2), 480–491.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

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

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

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

Appendix

4.A Detailed Results for the GVRP Instances

Detailed instance-by-instance results for the GVRP instance set are provided in Tables 4.5–4.8. The instance is described by the number of customers n, the number of vehicles k in the original CVRP instance, the number of clusters N, and the number of vehicles m. In addition, BKS gives the best known solution (written in bold if proven optimal) and *First found by* refers to the article (or the LMNS of this chapter or the branch-and-price (B&P) algorithm of the previous chapter) that has found this solution first. For our LMNS, we show the best solution out of ten runs (Best), the average solution over ten runs (Avg.), and the average total time over ten runs T derived by setting $\text{LMNS}^5_{10\,000}$ (which means the LMNS is run for 10 000 iterations and with $k_{\text{post}} = 5$). If the BKS was first found by our LMNS, we omit the number of iterations in the column *First found by* for simplicity. For example, we refer to setting $\text{LMNS}^5_{10\,000}$ by LMNS^5. If it was found with both $k_{\text{post}} = 3$ and $k_{\text{post}} = 5$ we state $\text{LMNS}^{3/5}$. Furthermore, LMNS* declares a solution found during computational experiments.

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

Table 4.5: Detailed results for the `GVRP-2` instances, subsets `A` and `B`.

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

Table 4.6: Detailed results for the `GVRP-2` instances, subsets `P` and `GC`.

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

Table 4.7: Detailed results for the `GVRP-3` instances, subsets `A` and `B`.

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

Table 4.8: Detailed results for the `GVRP-3` instances, subsets `P` and `GC`.

4.B Detailed Results for the `Golden` Instances

Analogous to Section 4.A, detailed results for the `Golden` instances are given in Tables 4.9 to 4.12 (without the number of vehicles k in the original CVRP instance). In addition, we give the best and average solution over ten runs for setting $\text{LMNS}^5_{10\,000}(10s)$, where the LMNS is stopped after the time limit of 10 seconds.

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

Table 4.9: Detailed results for the Golden instances 1-5.

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

Table 4.10: Detailed results for the Golden instances 6-10.

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

Table 4.11: Detailed results for the Golden instances 11-15.

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

Table 4.12: Detailed results for the Golden instances 16-20.

4.C Detailed Results for the `Li` Instances

Analogous to Section 4.A, detailed results for the `Li` instances are given in Table 4.13 (without the number of vehicles k in the original CVRP instance).

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

Table 4.13: Detailed results for the `Li` instances.

Chapter 5

Conclusion

The main goal of this thesis was to contribute to both versions of the clustered vehicle-routing problem, which both arise in real-life scenarios like courier companies, by presenting new exact and heuristic solution approaches. In both variants, customers are partitioned into clusters and customers of the same cluster have to be served by the same vehicle. The clustered vehicle-routing problem (CluVRP) requires consecutive visits to customers of the same cluster, while the soft-clustered vehicle-routing problem (SoftCluVRP) relaxes this constraint.

We proposed two new LMNS metaheuristics, one for each of the two problem variants, and different branch-and-price algorithms as the first exact solution methods for the SoftCluVRP. In the following, we summarize the main results chapter-wise and give concluding remarks.

In Chapter 2, we tackled the CluVRP with a new large multiple neighborhood search (LMNS) that combines using multiple destroy and repair operators together with searching several neighborhoods for post-optimization. All components work on a cluster level and exploit the cluster structure. A major novelty is a generalization of the Balas-Simonetti neighborhood that is able to decide on the best permutation of the clusters in a route and the routing through each cluster simultaneously. We analyzed the destroy and repair operators and the generalized Balas-Simonetti neighborhood by extensive computational experiments, showing that none of them is dispensable. The results reveal that our clearly structured LMNS is (more than) competitive with state-of-the-art solution methods. Furthermore, it is able to produce a couple of new best solutions for benchmark instances.

Chapter 3 and 4 addressed the SoftCluVRP that has been considered by only one article before. In Chapter 3, we designed and analyzed different branch-and-price algorithms for the exact solution of the SoftCluVRP. They differ in the solution method for the subproblem (which is a variant of the shortest-path problem with resource constraints) and we presented (i) sophisticated dynamic-programming based labeling algorithms and (ii) a branch-and-cut algorithm that works directly on the SoftCluVRP subproblem formulation. For this variant of the vehicle-routing problem, the integer programming-based branch-and-cut strongly outperforms the

fully-fledged labeling algorithm. The largest SoftCluVRP instances that we solve to optimality comprise more than 400 customers or more than 50 clusters. Future research may focus on further enhancing the branch-and-cut with new classes of valid inequalities as well as effective and fast separation heuristics. Furthermore, by comparing results for medium-sized instances to results for the CluVRP (Battarra *et al.*, 2014a), costs of optimal solutions were reduced by 6.21 % on average if the SoftCluVRP was considered. Hence, in practical applications this decision should be taken with care, if possible.

Chapter 4 provided an adaption of the aforementioned LMNS, tailored to the SoftCluVRP. New variants of the different components were suggested and a second post-optimization stage was added, working on single routes. The usefulness of the operators and both post-optimization components was shown by comprehensive computational studies. We produced new best solutions for more than half of the medium-sized benchmark instances and our LMNS is clearly superior to the only metaheuristic for the SoftCluVRP from the literature (Defryn and Sörensen, 2017). By comparing to lower bounds that were generated in Chapter 3, we could prove 13 new solutions to be the optimal solution.

For future research, soft-cluster constraints in districting and capacitated arc-routing applications (Butsch *et al.*, 2014; Belenguer *et al.*, 2014) could be an interesting avenue.

Bibliography

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, **19**(6), 621–636.

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

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

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., and Vance, P. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, **46**(3), 316–329.

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

Battarra, M., Cordeau, J.-F., and Iori, M. (2014b). *Pickup-and-Delivery Problems for Goods Transportation*, chapter 6, pages 161–191. Volume 18 of Toth and Vigo (2014).

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.

Belenguer, J. M., Benavent, E., and Irnich, S. (2014). The capacitated arc routing problem: Exact algorithms. In *Arc Routing*, chapter 9, pages 183–221. Society for Industrial & Applied Mathematics (SIAM).

Bentley, J. (1992). Fast algorithms for geometric traveling salesman problems. *Operations Research Society of America*, **4**(4), 387–411.

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

Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **26**, 1124–1137.

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

Cherkesly, M., Desaulniers, G., and Laporte, G. (2015). Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading. *Transportation Science*, **49**(4), 752–766.

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

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

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

Dakin, R. J. (1965). A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, **8**(3), 250–255.

Dantzig, G. and Ramser, J. (1959). The truck dispatching problem. *Management Science*, **6**, 80–91.

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

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, **1**, 3–18.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desaulniers, G., Pecin, D., and Contardo, C. (2017). Selective pricing in branch-price-and-cut algorithms for vehicle routing. *EURO Journal on Transportation and Logistics*, **8**(2), 147–168.

Drexl, M. and Irnich, S. (2012). Solving elementary shortest-path problems as mixed-integer programs. *OR Spectrum*, **36**(2), 281–296.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, **42**(5), 977–978.

Dumas, Y., Desrosiers, J., and Soumis, F. (1991). The pick-up and delivery problem with time windows. *European Journal of Operational Research*, **54**, 7–22.

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

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

Feillet, D., Dejax, P., Gendreau, M., and Guéguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Feillet, D., Dejax, P., and Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science*, **39**(2), 188–205.

Finner, H. (1993). On a monotonicity problem in step-down multiple test procedures. *Journal of the American Statistical Association*, **88**(423), 920–923.

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

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, **32**(200), 675–701.

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.

García, S., Fernández, A., Luengo, J., and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, **180**(10), 2044–2064. Special Issue on Intelligent Distributed Information Systems.

Ghiani, G. and Improta, G. (2000). An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research*, **122**, 11–17.

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**, 169–179.

Goeke, D., Gschwind, T., and Schneider, M. (2019). Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier. *Discrete Applied Mathematics*, **264**, 43–61.

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

Gschwind, T. and Drexl, M. (2019). Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, **53**(2), 480–491.

Gschwind, T., Irnich, S., Rothenbächer, A.-K., and Tilk, C. (2017). Bidirectional labeling in column-generation algorithms for pickup-and-delivery problems. Technical Report LM-2017-03, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

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

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

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, **130**(1), 449–467.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann.

Iman, R. L. and Davenport, J. M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in Statistics - Theory and Methods*, **9**(6), 571–595.

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

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Irnich, S. and Villeneuve, D. (2006). The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, **18**(3), 391–406.

Irnich, S., Funke, B., and Grünert, T. (2006). Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, **33**(8), 2405–2429.

Irnich, S., Toth, P., and Vigo, D. (2014). *The Family of Vehicle Routing Problems*, chapter 1, pages 1–33. Volume 18 of Toth and Vigo (2014).

Jepsen, M., Petersen, B., and Spoorendonk, S. (2008). A branch-and-cut algorithm for the elementary shortest path problem with a capacity constraint. Technical Report 08/01, Dept. of Computer Science, University of Copenhagen, Denmark.

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. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, **21**, 498–516.

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

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.

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

Parragh, S., Doerner, K., and Hartl, R. (2008). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, **58**(1), 21–51.

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

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

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

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

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

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, **51**(3), 155–170.

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

Ropke, S. and Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, **43**(3), 267–286.

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

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

Ryan, D. and Foster, B. (1981). An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, chapter 17, pages 269–280. Elsevier, North-Holland.

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.

Shaffer, J. P. (1986). Modified sequentially rejective multiple test procedures. *Journal of the American Statistical Association*, **81**(395), 826–831.

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. Cunningham, S. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 316–329. Springer Berlin Heidelberg.

Tarjan, R. E. (1979). A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, **18**(2), 110 – 127.

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

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

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

Vidal, T. (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.

Vidal, T., Battarra, M., Subramanian, A., and Erdoğan, G. (2017). Corrigendum to "Hybrid metaheuristics for the clustered vehicle routing problem [Comput. Oper. Res., 58 (2015): 87-99]". *Computers & Operations Research*, **85**(Supplement C), 206–207.

Wolsey, L. (1998). *Integer Programming*. Wiley, Chichester, New York.