In-depth analysis of granular local search for capacitated vehicle routing

Working Paper DPO-2020-03 (version 1, 20.03.2020)

Christian Schröder, Michael Schneider, Jean Bertrand Gauthier, and Timo Gschwind

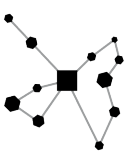
{schroeder|schneider}@dpo.rwth-aachen.de
Deutsche Post Chair – Optimization of Distribution Networks
RWTH Aachen University, Germany

{jgauthie|gschwind}@uni-mainz.de
Chair of Logistics Management
Johannes Gutenberg University Mainz, Germany

Abstract

Local search (LS) belongs to the core components of most state-of-the-art metaheuristics for vehicle routing problems (VRPs). Over the last decades, many variants of LS using different neighborhood operators, exploration strategies, and speedup techniques have been developed. These design choices can critically influence the performance of a LS in terms of both solution quality and computational effort. Despite the importance of LS in metaheuristics for VRPs, systematic investigations on the impact and importance of different design decisions in LS are nonexistent even for basic VRP variants, and clear recommendations on meaningful combinations of these decisions are not available. In this paper, we systematically study the impact of design decisions in a granular LS for the capacitated VRP (CVRP). To this end, we compare the performance of a large number of algorithmic variants of the granular LS on two CVRP benchmark sets. We use a Wilcoxon signed-rank test to determine non-dominated algorithmic variants, and we use performance profiles to visualize the results with regards to solution quality and runtime. In this way, we are able to identify good combinations of decisions and give final design recommendations on granular LS. Although our computational analysis is limited to the CVRP, we expect our findings to be transferable to other VRP variants to a large extent.

Keywords: *vehicle routing, granular local search, statistical analysis*



Deutsche Post
Chair - Optimization of
Distribution Networks

RWTH AACHEN
UNIVERSITY

1 Introduction

The family of vehicle routing problems (VRPs) is one of the most widely studied classes of combinatorial optimization problems. Its archetypical variant, the classical capacitated VRP (CVRP), can be defined on a complete directed graph $G = (\mathcal{V}, \mathcal{A})$. The vertex set is given by $\mathcal{V} = 0 \cup \mathcal{V}_c$, where 0 is the depot and $\mathcal{V}_c$ is the set of customers. The set of arcs $\mathcal{A} = \mathcal{A}_c \cup \mathcal{A}_d$ is the union of the customer arc set $\mathcal{A}_c = \{(i,j) \mid i,j \in \mathcal{V}_c, i \neq j\}$ and the depot arc set $\mathcal{A}_d = \{(i,j) \mid i \in \mathcal{V}_c, j = 0 \vee i = 0, j \in \mathcal{V}_c\}$. A cost c_{ij} and a travel time t_{ij} are associated with each arc $(i,j) \in \mathcal{A}$. A nonnegative demand q_i is associated with each vertex $i \in \mathcal{V}$ ($q_0 = 0$ for the depot). A homogeneous fleet of K vehicles with capacity Q is located at the depot to perform the customer visits. A solution to the CVRP is a set of vehicle routes, such that each customer is visited exactly once and the capacity limits of the vehicles are respected. It is optimal if it is cost-minimal.

The great success of VRPs as a research topic stems from the fact that they are both theoretically challenging (they are $\mathcal{NP}$-hard) and of high relevance in transportation planning (see, e.g. Toth and Vigo 2014)—both as standalone problems and as part of integrated problems like in location routing, inventory routing, and production routing. Realistically-sized instances of even basic VRP variants can typically not be solved to proven optimality in reasonable time. State-of-the-art exact solution approaches for basic VRPs, which are very often based on branch-price-and-cut (see Costa et al. 2019), in general already exhibit long and unpredictable computation times for instances of 50–200 customers. This leaves heuristics as the only viable alternative for solving larger VRP instances with fast response times. Local search (LS) is a simple yet successful heuristic method that has been applied to many combinatorial optimization problems. LS methods iteratively and systematically modify a given solution in order to reach better solutions. In early research on VRPs, LS was used as a standalone method to tackle traveling salesman problems (TSPs) and VRPs (e.g., Lin 1965, Kindervater and Savelsbergh 1997), but these pure LS implementations are no longer state of the art. However, successful metaheuristics feature diversification and intensification techniques: diversification mechanisms try to ensure that the metaheuristic search is guided to explore large parts of the solution space and to overcome local optima, intensification components focus on an in-depth exploration of promising areas of the solution space. LS remains important as an extremely popular intensification component in metaheuristics for VRPs.

Independent of their usage, as standalone methods or as components of metaheuristic algorithms, LS methods must be realized in the most efficient manner possible, i.e., speedup techniques to accelerate the search of the neighborhood have to be used. One possibility to speed up the search is to find an intelligent order in which the neighborhood moves are evaluated. Examples of such techniques are lexicographic search (Savelsbergh 1990) and sequential search (Irnich et al. 2006). Second, so-called locality tracking methods make use of the fact that LS moves only affect small parts of the solution, and therefore it is not necessary to reevaluate the large majority of other potential moves after a certain move has been applied (see, e.g., Beek et al. 2018). The third possibility is the pruning of neighborhoods, i.e., to heuristically reduce the size of the neighborhoods. A very popular example is granular search—an implementation of the candidate list strategy—introduced in Toth and Vigo (2003). Granular search has successfully been applied within both standalone LS (Moretti Branchini et al. 2009) and metaheuristic approaches like tabu search (TS, Toth and Vigo 2003, Schneider et al. 2017), genetic algorithms (Vidal et al. 2012), and variable neighborhood search (VNS, Labadie et al. 2012, Escobar et al. 2014a).

The concrete definition of a basic LS requires many different, potentially dependent design choices like the

selection of neighborhood operators, the use of speedup techniques (e.g., granular search), or the pivoting rule (e.g., first vs. best improvement). These design choices can critically influence the performance a LS in terms of both solution quality and computational effort. Surprisingly, systematic investigations on the impact and importance of different design decisions are not existent even for basic VRP variants, and recommendations on meaningful combinations of these decisions are not available. In this paper, we systematically study the impact of design decisions in a granular LS (GLS) for the CVRP. To this end, we compare the performance of a large number of algorithmic variants of the GLS on two CVRP benchmark sets. Using statistical tests, we identify good combinations of decisions and give final design recommendations for GLS. Although our computational analysis is limited to the CVRP as the most basic VRP variant, we expect our findings to transfer to a large part also to VRP variants including additional constraints.

The remainder of the paper is organized as follows. Section 2 provides an overview of the related literature. Section 3 gives a detailed description of GLS and its components, and explains the different design decisions. Section 4 describes our test design, the techniques that we use to assess the performance of the algorithmic variants, and our extensive computational studies on the performance of the algorithmic variants together with clear design recommendations. Section 5 concludes the paper.

2 Related literature

Hansen and Mladenović (2006) examine and compare the performance of first and best improvement as pivoting rules for a 2-opt LS for the TSP. The authors find that for random starting solutions, best improvement provides worse results than first improvement. When using starting solutions constructed by a greedy or nearest neighbor heuristic, however, best improvement is better and faster on average. Mjirda et al. (2017) study the behavior of several sequential variable neighborhood descent (VND) strategies for the TSP (standalone and within a VNS), using the neighborhood operators 2-opt, relocate, and string-relocate with strings of length two (see also Section 3.1). The authors find that using a construction heuristic is superior to starting from random solutions and that the behavior of first and best improvement as pivoting rules is the same as reported in Hansen and Mladenović (2006).

Arnold and Sörensen (2019) study different aspects of LS within a guided local search for the CVRP. Besides analyzing several design choices related to the guided local search framework, they compare different neighborhoods and pruning strategies for the LS. The authors observe that larger neighborhoods help to find better solutions and that smaller neighborhoods do not find improving moves faster than larger neighborhoods even at the beginning of the search. This behavior is attributed to the effective pruning of the neighborhoods.

Finally, Lü et al. (2011) study LS components for the curriculum-based course timetabling problem, both as standalone method and within different metaheuristic frameworks. The authors experimentally analyze the characteristics of different neighborhoods and their combinations by evaluating their search capability using the criteria: percentage of improving neighbors, improvement strength, and number of search steps.

3 Designing granular search for the capacitated VRP

LS improves a given solution by iteratively applying small local changes, so-called moves, until a local optimum with respect to the considered neighborhood operators (see Section 3.1) is reached. To speed up

the search, we do not search entire neighborhoods but instead use granularization to reduce the size of the neighborhoods to traverse (Section 3.2). The concrete strategy to explore the given neighborhoods is given by (i) the pivoting rule used to define the search for an improving neighbor within the search neighborhood, and (ii) the definition of the complete search neighborhood based on the selection and order of applying the neighborhood operators (Section 3.3).

3.1 Neighborhood operators

Given a current solution x and a list of neighborhood operators $\mathcal{N}$, each individual neighborhood operator (or simply neighborhood) $\eta \in \mathcal{N}$ generates a set of neighboring solutions $N_\eta(x)$. With slight abuse of notation, we use the $\in$-operator also to denote that an entry is contained in a list. The list of neighborhood operators $\mathcal{N}$ used to define the complete search neighborhood is a crucial design decision in LS because the probability of finding neighboring solutions with larger gains increases with each additional neighborhood operator, but this comes at the cost of exploring the additional neighborhood.

Funke et al. (2005) provide a detailed overview and analysis of the most relevant neighborhood operators for solving VRPs. We restrict our analysis to the following seven operators, whose size is quadratic in the number of vertices of the given problem instance (abbreviations of the operator names are given in parentheses).

2-opt (2) is used in intraroute fashion only and removes two non-consecutive arcs from a route, inverts the vertex segment between the removed arcs, and inserts two new arcs to regenerate a feasible route.

2-opt* (2*) is used in interroute fashion only and removes one arc from each route and reconnects the first part of the first route with the second part of the second route and vice versa.

relocate (rel) moves a vertex from its current position to a different one in the same (intraroute) or in a different route (interroute).

exchange (ex) swaps the position of two vertices within the same or between different routes.

string-relocate (strel) relocates a string of up to ℓ_r contiguous vertices within the same route or between different routes. To restrict the search effort, we limit ourselves to $\ell_r = 3$.

string-exchange (stex) is used in intra- and interroute fashion and exchanges two nonoverlapping vertex strings of up to ℓ_e vertices each. To omit double-checking of exchange and string-relocate moves, we impose that both strings are nonempty and together they cover at least three vertices. We thus evaluate $\ell_e^2 - 1$ combinations of string lengths. To limit the search effort, we use $\ell_e = 3$.

string-exchange-inverted (stexi) is also used in intra- and interroute fashion and differs from string-exchange in the fact that both strings are inverted before they are reinserted. We again impose that both strings are nonempty and together cover at least three vertices, thus evaluating $\ell_i^2 - 1$ combinations of string lengths. To limit the search effort, we use $\ell_i = 3$.

More detailed information on the above operators can, e.g., be found in Funke et al. (2005).

3.2 Granularization

In granular search (Toth and Vigo 2003), neighborhoods are traversed based on a so-called list of generator arcs $\mathcal{A}_g$. A generator arc $(i, j) \in \mathcal{A}_g$ in combination with a neighborhood operator $\eta \in \mathcal{N}$ (and the length(s) of the string(s) in case of the string operators introduced above) uniquely identifies a move. More precisely, the generator arc (i, j) modifies the solution in a manner that after application of the move, vertex i is followed by vertex j in the resulting solution, and all other arcs to be removed and added are specified. Note that this does not mean that only generator arcs can be inserted because all neighborhood operators insert more than one arc into the solution.

To speed up the neighborhood traversal, we use granular neighborhoods, i.e., we reduce the size of a neighborhood by only keeping promising arcs in the list of generator arcs $\mathcal{A}_g$. A so-called sparsification criterion (e.g., arc length) is used to sort the arcs contained in the original arc set, and a sparsification factor π determines the percentage of the sorted arcs to keep. For instance, we could keep the 20% of shortest feasible arcs in a CVRP problem instance as generator arcs. Given the current solution x, the granular neighborhood $N_\eta(x, \mathcal{A}_g(\pi))$ is the set of all solutions that is generated by neighborhood operator $\eta \in \mathcal{N}$ with the sorted and truncated list of generator arcs $\mathcal{A}_g(\pi)$.

Dynamic sparsification, i.e., adapting the strength of sparsification during the search, has been successfully used in several applications of granular search, e.g., in Labadie et al. (2012) for an orienteering problem, or in Escobar et al. (2014b) for the multidepot VRP. We realize dynamic sparsification by means of a list Π containing multiple sparsification factors in increasing order. If in one iteration of the GLS we are not able to find an improving move after traversing all neighborhoods, we try with the next sparsification factor in Π. This process is repeated until either an improving solution can be found, or we end up with a local optimum after searching with the largest $\pi \in \Pi$. If an improving solution is found, the next iteration of the GLS again starts with the smallest $\pi \in \Pi$.

In addition, a separate treatment of customer arcs $\mathcal{A}_c$, i.e., arcs connecting two customers, and depot arcs $\mathcal{A}_d$, i.e., arcs connecting a depot with a customer, proved to be clearly beneficial in the context of a granular TS for the VRP with time windows (Schneider et al. 2017). We implement this feature as follows: We keep two separate lists, one for the customer arcs and one for the depot arcs. Each time a generator arc list is created, both lists are sorted independently according to the sparsification criterion, and then, they are truncated using the same sparsification factor π. Finally, both lists are merged, and the resulting list is sorted according to the sparsification criterion, yielding $\mathcal{A}_g(\pi)$.

3.3 Exploration strategy

The strategy to explore the neighborhoods described above is determined by the selected pivoting rule (Section 3.3.1) and the way in which the neighborhoods are traversed in the complete search neighborhood (Section 3.3.2).

3.3.1 Pivoting rules

In each iteration of the GLS, the goal is to find a neighbor within the complete search neighborhood that is admissible with regards to the pivoting rule. We investigate the following five well-known pivoting rules:

first improvement (FI): Apply the first move that leads to a solution with better objective function value.

k-first improvement (k-FI): Search until k improving moves have been found and apply the one with the strongest improvement.

randomized-k-first improvement (r-k-FI): In each iteration, draw a uniformly distributed random integer $u \in [1, k]$, then use u-first improvement.

k-sequential improvement (k-SI): Search until k new best moves have been found and apply the k-th. Note that, in contrast to the previous strategies, a sequentially improving move does not only need to improve on the current solution x, but also needs to improve on the best move found in the iteration so far.

best improvement (BI): Apply the best move found.

3.3.2 Definition of complete search neighborhood

We investigate two frequently used schemes to define the complete search neighborhood of each iteration of our GLS, namely as composite neighborhood over all neighborhood operators or as a variable neighborhood descent (VND) on a sorted list of the neighborhood operators.

Composite neighborhood: In each iteration, the search considers the union of all neighborhood operators, i.e., $\bigcup_{\eta \in \mathcal{N}} N_\eta(x, \mathcal{A}_g(\pi))$. To prevent biasing the search towards moves found by certain neighborhood operators, operators are not completely searched before switching to the next one, but the generator arc list is traversed in sorted order, and for each generator arc $(i, j) \in \mathcal{A}_g(\pi)$, the move defined by each neighborhood operator $\eta \in \mathcal{N}$ (traversed in the order given in Section 3.1) is evaluated.

VND: In each iteration, the neighborhood operators are traversed according to increasing complexity, which corresponds to the order given in Section 3.1. Contrary to the composite neighborhood, VND always completely searches all moves of a specific neighborhood $N_\eta(x, \mathcal{A}_g(\pi))$ before moving to the next neighborhood, i.e., for each operator $\eta \in \mathcal{N}$, all possible moves $\{((i, j), \eta, x),\ (i, j) \in \mathcal{A}_g(\pi)\}$ are evaluated in the order given by the generator arc list $\mathcal{A}_g(\pi)$. If at least one improving neighbor is found in a certain neighborhood, no additional neighborhoods are searched, i.e., the search continues with the next iteration. This is also true if the improving neighbor is not admissible with regards to the pivoting rule. Thus, even if the pivoting rule aims to find additional improving neighbors, more complex neighborhood operators are only evaluated if no improving solution has been found.

Algorithms 1 and 2 show pseudocodes of one local search iteration using a composite neighborhood and a VND, respectively. For-loops over the entries of a list, e.g., *for* $\eta \in \mathcal{N}$, are assumed to iterate over the entries of a list in the given order. The counters r_{imp} and r_{simp} keep track of the numbers of identified improving and sequentially improving moves. The method pivotingRule(r_{imp}, r_{simp}) determines whether an admissible neighbor according to the pivoting rule has been found.

4 Computational studies

In this section, we present our experimental analysis of the different GLS variants. Section 4.1 describes the benchmark instances and computing environment. In Section 4.2, we detail the test design, i.e., which

Algorithm 1: composite neighborhood

```
input: current solution x
for π ∈ Π
    r_imp, r_simp ← 0
    x_best ← ∅; c(x_best) ← ∞
    for (i, j) ∈ A_g(π)
        for η ∈ N
            x' ← move((i, j), η, x)
            if c(x') < c(x)
                r_imp ← r_imp + 1
                if c(x') < c(x_best)
                    x_best ← x'
                    r_simp ← r_simp + 1
                if pivotingRule(r_imp, r_simp)
                    return x_best
    if x_best ≠ ∅
        return x_best
return ∅
```

Algorithm 2: VND

```
input: current solution x
for π ∈ Π
    for η ∈ N
        r_imp, r_simp ← 0
        x_best ← ∅; c(x_best) ← ∞
        for (i, j) ∈ A_g(π)
            x' ← move((i, j), η, x)
            if c(x') < c(x)
                r_imp ← r_imp + 1
                if c(x') < c(x_best)
                    x_best ← x'
                    r_simp ← r_simp + 1
                if pivotingRule(r_imp, r_simp)
                    return x_best
        if x_best ≠ ∅
            return x_best
return ∅
```

combinations of algorithmic features are investigated. Section 4.3 explains how the performance of different algorithmic variants is evaluated. The main insights for the design of GLS methods for the CVRP are discussed in Section 4.4.

4.1 Benchmark instances and computational environment

We use a subset of the CVRP benchmark instances of Irnich et al. (2006) (IFG) and the entire benchmark set of Uchoa et al. (2017) (UPPPVS). The IFG set comprises a total of 600 instances, which are grouped into ten series that differ with regards to the coefficient of variation of the demand distribution. To keep the computational effort in a reasonable range, we restrict ourselves to the first 36 instances of series 1, which contain a central depot and between 250–1000 customers that are uniformly distributed in the plane. For each instance size, there are four different demand distributions, generated as follows: customer demands are randomly drawn from a uniform distribution over $[10, 30]$, and the vehicle capacity is determined by multiplying the average demand by factors of $\lambda = 25, 50, 75$, and 100. This entails that near-optimal solutions should have an average number of roughly λ customers on each tour.

The UPPPVS set comprises 100 instances that contain between 100 and 1000 customers. Customers are positioned either randomly, clustered, or random-clustered, and the depot is located either in the center, the bottom left corner, or at a random point. Seven different types of demand distributions are used that cover different distributions of the total demand and different coefficients of variation. Vehicle capacities are determined such that very short, short, medium, long, and very long routes emerge.

The GLS algorithm is implemented in C++ and compiled into 64-bit single-thread code using MS Visual Studio 2017. All experiments are performed on the c18m partition of the RWTH Aachen computer cluster, which contains 1032 nodes of the type Intel HNS2600BPB. Every node has 192 GB of memory and contains two sockets which in turn contain 24 cores each. All cores are clocked at 2.1 GHz.

4.2 Test design

4.2.1 Feature combinations

We explain how we obtain the final algorithmic variants by making the design choices described in Section 3.

Neighborhood operators: Considering all possible subsets of the seven neighborhood operators described in Section 3.1 would lead to $2^7 = 128$ possibilities. To keep the number of algorithmic variants reasonable, we limit the study to the following 17 neighborhood operator subsets. The first subset contains the four basic operators (2, 2*, rel, ex). Twelve subsets are generated by taking all possibilities of drawing three of the four basic operators and adding one string-operator (strel, stex, stexi). Three subsets contain all four basic operators and one string-operator, and finally one set contains all operators. A detailed list of the considered sets of neighborhood operators is given in the appendix.

Granularization: Given the convincing results obtained in Toth and Vigo (2003), we use arc length as sparsification criterion in all algorithmic variants. In addition, all variants use separate lists for depot and customer arcs as explained above. Concerning the dynamic sparsification, we consider the following three variants that differ in the strength of sparsification (defined by their respective Π vector):

- strong: $\Pi = (0.025, 0.05, 0.1)$
- medium: $\Pi = (0.05, 0.1, 0.2)$
- weak: $\Pi = (0.075, 0.15, 0.3)$

Pivoting rules: We investigate the five pivoting rules FI, k-FI, r-k-FI, k-SI, and BI using $k = 10$ for k-FI and r-k-FI and $k = 3$ for k-SI in our experiments.

Definition of complete search neighborhood: A composite neighborhood definition as well as a VND setting are tested.

An (algorithmic) variant ω is defined by selecting one of the predefined values for each of the features listed above. Let Ω denote the set of all variants. Because no incompatible settings exist, we have $|\Omega| = 17 \times 3 \times 5 \times 2 = 510$ different variants. Each of these variants is run on each of the problem instances described in Section 4.1 starting from 5000 initial solutions that are generated as described in the following section.

4.2.2 Construction heuristics

We aim to generate a set of initial solutions that is as diverse as possible to cover a broad range of the search space. To this end, we use five randomized construction heuristics (nearest neighbor, nearest insertion, random insertion, accelerated best insertion, savings) and generate 1000 different and feasible initial solutions with each of them. Nearest neighbor is the only sequential approach, the remaining four are parallel. Details about the construction heuristics are provided in the following paragraphs.

Nearest neighbor We randomly select one of the five customers that are closest to the last customer added to the solution, and we either connect the two (if this is capacity feasible), or we open a new route containing only the selected customer if this leads to a smaller cost increase. In the latter case, we close the previous route. We start with the customer with the smallest index and repeat the process until all customers are routed.

Insertion heuristics In the three insertion heuristics, we select a certain set of unrouted customers in each iteration. For each of them, we consider the insertion moves that either open a new route containing only this customer or that insert the customer into any existing route in capacity-feasible fashion. We order these moves (over all the unrouted customers to be inserted) according to cost and randomly select the insertion to carry out from the five most cost-effective ones (if less than five possibilities exist, we randomly select among the remaining moves). The process is repeated until all customers are included in the solution.

The three variants of the insertion heuristic differ with regards to how the set of unrouted customers to be inserted is selected in each iteration:

- *Random:* only a single customer to be inserted is randomly selected from the yet unrouted customers.
- *Nearest:* starts with the customer with the smallest index in the first iteration, and afterwards a random one of the five unrouted customers with the minimum distance to any of the routed customers is chosen.
- *Accelerated best:* in the first iteration, the customer with the minimal distance to the depot is inserted. Afterwards, not only a single customer is considered for insertion, but all unrouted customers within a certain proximity of the already routed customers are taken into consideration. There is of course a tradeoff between the number of considered customers and the runtime. Preliminary tests showed that considering the 20% nearest customers of each routed customer leads to good results within reasonable runtimes.

Savings The savings heuristic iteratively merges routes, starting from a set of dedicated routes each serving a single customer. A merge connects the last customer of a route with the first customer of another route, or vice versa. We obtain randomized solutions by first sorting the savings in decreasing order and then shuffling items as follows: Starting from the largest saving, with a probability of 0.3, each list entry is swapped with one of the five subsequent entries selected at random. Then, the merges are performed in the order given by the shuffled list. Merges are only performed if they respect vehicle capacities.

4.3 Performance assessment

Our aim is to compare the performance of heuristic algorithms, but in general, the considered algorithmic variants will generate different local optima within different runtimes for each problem instance. Our approach to deal with the underlying tradeoff between solution quality and runtime is to get rid of the quality dimension by defining that a certain variant *solves* an instance if it reaches, i.e., matches or beats, a certain target quality on this instance. In this fashion, we use the information on how often a variant solves an instance for all starting solutions and the time it takes to perform the search runs to compute an average *solution time* for an instance that already comprises solution quality.

More precisely, given an instance to solve, for each pair of algorithmic variant and initial solution index (ω, s), GLS returns a local optimum with objective value $\tilde{z}_{\omega s}$ in $t_{\omega s}$ seconds. We compute the relative gap $\rho_{\omega s}$ to a target objective value $\hat{z}$ in percent, the latter given by the best objective value found over all variants and initial solutions for the given instance (i.e., $\hat{z} = \min_{(\omega,s)} \tilde{z}_{\omega s}$):

$$\rho_{\omega s} = ((\tilde{z}_{\omega s} - \hat{z})/\hat{z}) \cdot 100.$$

Based on the computed relative gaps and a maximum accepted deviation (target quality) $\gamma \geq 0$, let $C = \{s \mid \rho_{\omega s} \leq \gamma\}$ denote the set of initial solution indices for which a *successful run* occurs, i.e., for which we consider the instance as solved. We derive the average time to success of a variant on the given instance as

$$\bar{t}_\omega = \sum_{s=1}^{\max C} t_{\omega s} \,/\, |C|.$$

Note that by definition, we discard all runs after the last successful run to avoid dealing with a truncated observation, which would add little value to the measurements. If no successful run occurs, we set $\bar{t}_\omega = \infty$.

To compare the performance of the algorithmic variants, we follow the statistical analysis method of Coffin and Saltzman (2000). Instead of measuring the time required to reach optimality on a problem instance, a statistical unit now measures the average time to success as defined above. Let $\bar{\bar{t}}_\omega$ denote the mean of $\bar{t}_\omega$ over all considered instances. A variant ω_1 is considered better than an alternative ω_2 if the average time $\bar{t}_{\omega_1}$ is smaller than $\bar{t}_{\omega_2}$ in a consistent manner across the benchmark set. In statistical terms, the null hypothesis is that GLS behaves similarly with either variant, i.e., $H_0 : \bar{\bar{t}}_{\omega_1} = \bar{\bar{t}}_{\omega_2}$ whereas the alternative hypothesis states that an impact can be observed, i.e., $H_1 : \bar{\bar{t}}_{\omega_1} < \bar{\bar{t}}_{\omega_2}$. A variant ω_2 is dominated if evidence allows us to reject H_0 at a level of significance α. Non-dominated variants are tested pairwise until a Pareto set $\Omega^\star \subset \Omega$ remains. We compute the Pareto set using the Wilcoxon signed-rank test with $\alpha = 5\%$, and we set the target quality to $\gamma = 1\%$.

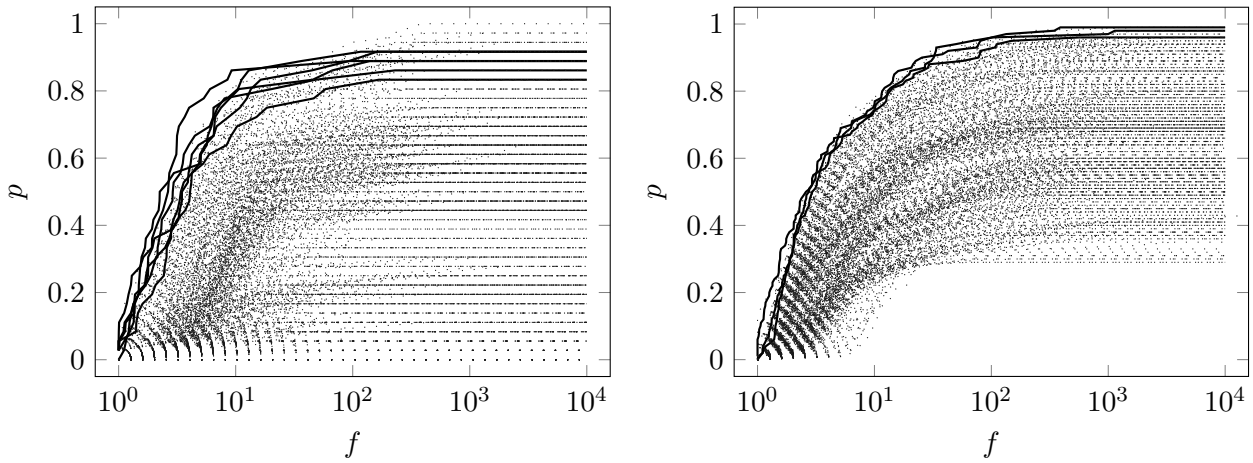
To visualize the performance of the algorithmic variants, we use a plot of performance profiles (Dolan and Moré 2002). In such a plot, each curve—in the first quadrant of a Cartesian coordinate system—represents the performance profile of one variant on the given instance set. A point (f, p) on a curve means that the associated variant can solve the proportion p of the instances when given f times the time taken by the fastest variant to solve each of the respective instances. For the special case of $f = 1$, each curve shows the proportion of instances for which the associated variant is the fastest of all variants considered. Because we are aware of the shortcomings of ranking algorithms based on a visual inspection of performance profiles (see Gould and Scott (2016)), we only use them as visualization tool. The ranking, i.e., the computation of the Pareto set of variants, is based on the statistical analysis described above.

Figure 1 depicts the plots of performance profiles for both benchmark sets. Non-dominated variants are shown using solid lines, dominated variants using dotted lines. As the figure shows, only six variants remain non-dominated for the IFG set, and only three for the UPPPVS set.

The plots also indicate that there is a correlation between the performance profiles and the statistical dominance tests. That is, the non-dominated variants perform well on average over the full benchmark sets (this is especially true for the UPPPVS instances): Their profiles are located rather at the top left over the complete plot meaning that they are among both those variants that solve the most instances, i.e., they reach a large value of p, and those variants that solve the largest fractions of the instances for any given time factor f. We

stress that this correlation does not always have to hold.

For example, in the most extreme case, a variant would also be non-dominated if it were very good (the best variant) for a handful of instances, but unable to solve any of the remaining instances. Such a variant has a performance profile that sharply goes upward and then remains parallel to the f-axis at some relatively small value p. The non-dominated curve that peaks at around $p = 80\%$ in Figure 1a is a mild example of such a situation. Even if one would probably discard this variant from the visual inspection of the performance profiles, it remains non-dominated because of its outstanding performance on a relatively small proportion of the instance set.



(a) IFG instance set

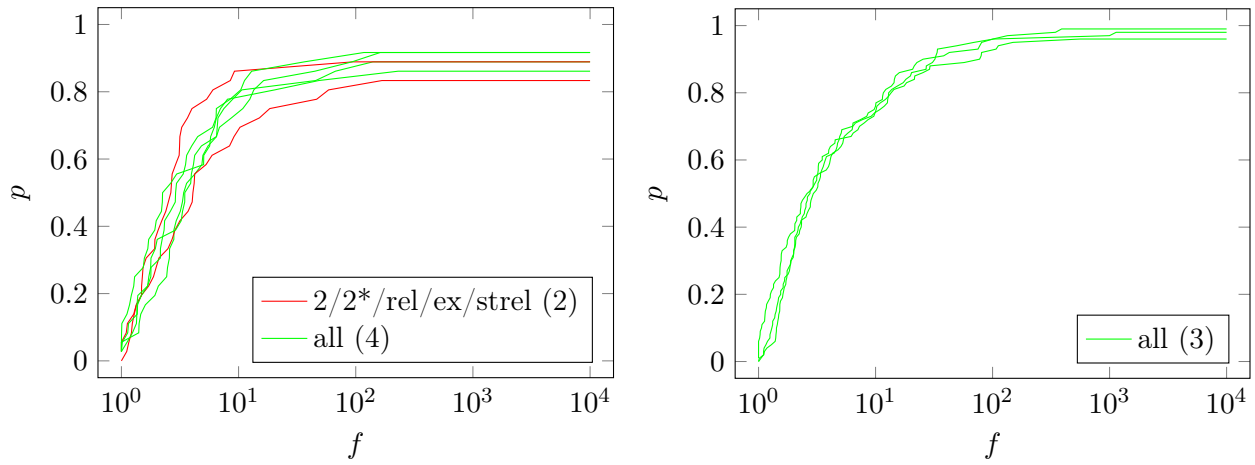
(b) UPPPVS instance set

Figure 1: Plot of performance profiles of all algorithmic variants on the CVRP benchmark instances.

4.4 Analysis of the Pareto set and design recommendations

To derive design recommendations for a (granular) LS for the CVRP, we analyze the non-dominated variants with regards to their algorithmic features. We focus on each design decision individually before investigating their combinations.

Neighborhood operators Figure 2 shows the performance profiles for the two instance sets, highlighting the selection of neighborhood operators. For IFG, four of the six non-dominated variants use all neighborhood operators, and two use the 5-operator subset 2/2*/rel/ex/strel. For UPPPVS, all non-dominated variants use all neighborhood operators. Remembering that our analysis takes into account the runtime required to search additional neighborhoods, the results are far from what could be called common knowledge in designing (granular) LS. Although there clearly exist some papers in which a relatively large number of neighborhood operators are used (see, e.g., Schneider and Löffler 2019), many VRP papers work with only three or four operators. Notably, several works explicitly aiming at solving VRP variants in short runtimes or on solving large-scale VRPs (e.g., Li et al. 2005, Schneider et al. 2017) make do with a rather small number of neighborhood operators. Our results indicate that this type of decision may not be chiseled in stone. A deeper analysis of the results obtained by the dominated variants shows that the worst variants are those that use only four neighborhood operators, especially if these four operators do not include relocate or string-relocate.



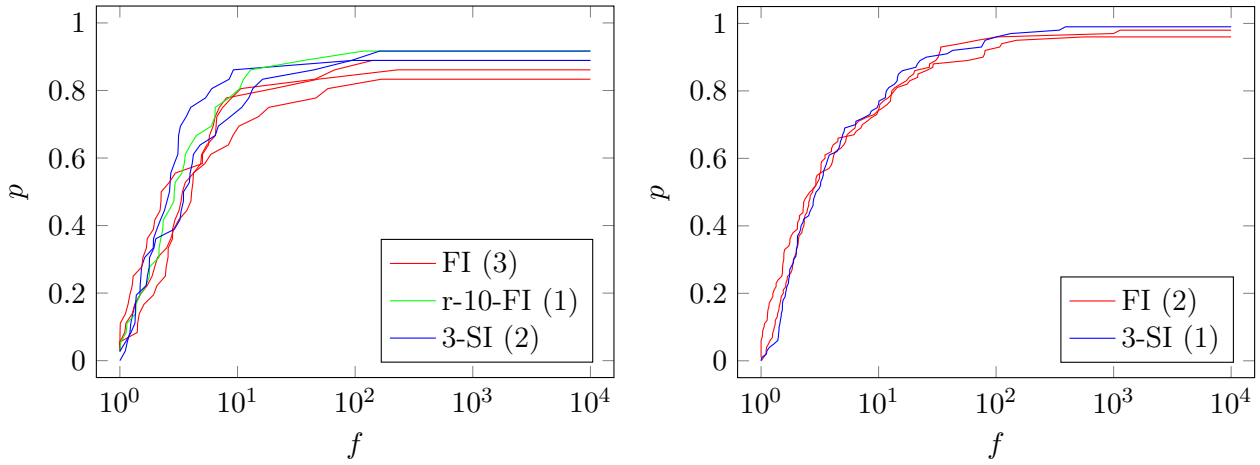
(a) IFG instance set

(b) UPPPVS instance set

Figure 2: Performance profiles of non-dominated variants grouped according to sets of neighborhood operators. In the legend, the number of occurrences of each subset is given in brackets.

Overall, our results indicate that a large number of neighborhood operators is beneficial to achieve a good tradeoff between solution quality and run-time. When working with a reduced number of neighborhood operators, it seems recommendable to include relocate and/or string-relocate.

Pivoting rules Figure 3 shows the performance profiles for the two instance sets putting the focus on the selection of the pivoting rule. Although no clear recommendation can be given, first improvement and



(a) IFG instance set

(b) UPPPVS instance set

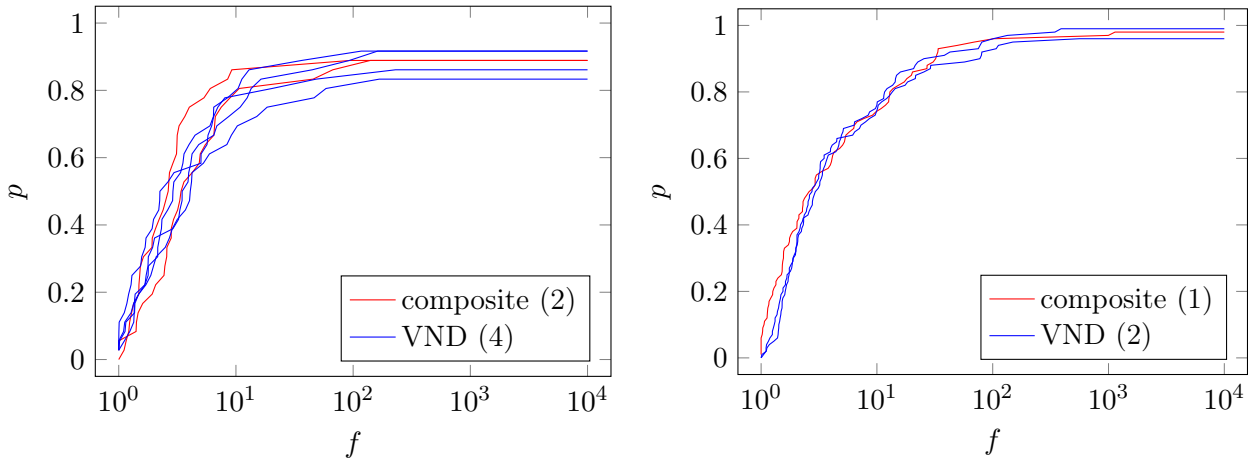
Figure 3: Performance profiles of non-dominated variants grouped according to selected pivoting rule.

sequential improvement are found superior to the other options. Interestingly, best improvement is not used in any of the non-dominated algorithmic variants. The superior behavior of first and sequential improvement over best improvement may be explained by the design of GLS: because the generator arc lists are sorted from most to least promising arc, we assume that a first improvement search returns moves that are often almost as good as the best moves but are found in much shorter time.

Our results are in stark contrast to common practice in the CVRP literature, where best improvement has been utilized in a fair share of papers. This decision should at least be critically questioned, especially when using

some type of guided search in which good moves are supposed to be found early and if best improvement is selected based on the gut feeling that this is the best strategy to put more emphasis on solution quality (remember that our analysis incorporates solution quality).

Definition of complete search neighborhood Figure 4 depicts the selection of search neighborhood definition in the non-dominated variants. While no definite recommendation can be given, the results are con-



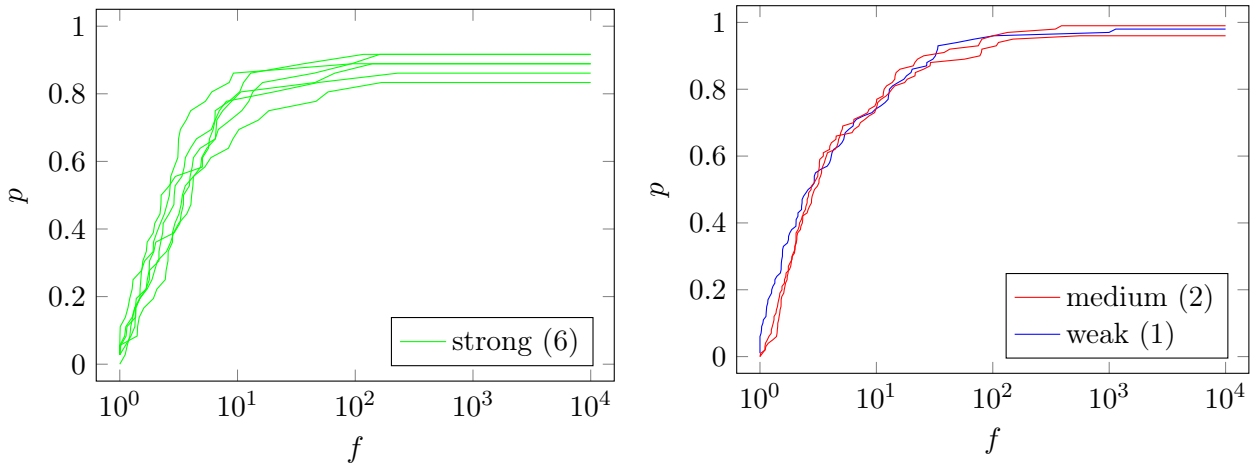
(a) IFG instance set

(b) UPPPVS instance set

Figure 4: Performance profiles of non-dominated variants grouped according to the definition of the search neighborhood.

sistent over the two instance sets and show that a VND scheme is present in the non-dominated variants twice as often as the composite neighborhood definition. Again, this result is not necessarily in line with the frequency in which the two types of neighborhood definitions have been used in CVRP papers in the past.

Granularization Figure 5 shows the selected sparsification strengths in the non-dominated variants. For



(a) IFG instance set

(b) UPPPVS instance set

Figure 5: Performance profiles of non-dominated variants grouped according to sparsification strengths.

the first time in our analysis, the results found for the two benchmark sets are not consistent: while on the IFG set, strong sparsification is clearly the dominant strategy, medium and even weak sparsification show

the best performance on the UPPPVS set. This indicates that instance characteristics have a significant influence on the choice of an accurate sparsification strength. Also note that the sparsification strength must always be analyzed in combination with the sparsification criterion used for sorting the generator arcs. In the following, we investigate the two instance sets with regards to several characteristics to explain why the design recommendations for the two sets differ. This also provides an example of the type of analysis that can be useful when deciding on granularization features (sparsification strength, sparsification criterion, treatment of customer and depot arcs) in a LS for the CVRP.

Several of the characteristics that we consider to distinguish the two instance sets are based on the best-known solutions (BKS) of the individual instances. We use the BKS of Gauthier and Irnich (2020) for IFG and those available in the accompanying `CVRPLIB` (http://vrp.galgos.inf.puc-rio.br/) web resource for UPPPVS.

Minimum sparsification factors: For each instance, we calculate the sparsification factor that is required to include all arcs of the BKS in the list of generator arcs. We are aware that a certain arc does not have to be included in the list of generator arcs to be inserted into the solution, but we still believe that this measure is a good indicator to show the differences between the two instance sets. Table 1 shows the average values computed for the depot arcs (π_d^{bks}) and the customer arcs (π_c^{bks}) of the two instance sets. The results show that the minimum sparsification factors are clearly higher for the UPPPVS instances.

	IFG	UPPPVS
π_d^{bks}	0.28	0.88
π_c^{bks}	0.09	0.34

Table 1: Minimum sparsification factors to include BKS depot and customer arcs in the list of generator arcs.

This indicates that longer arcs are part of high-quality solutions, which explains the usage of medium or weak sparsification in non-dominated variants. Moreover, we find that the required sparsification factors for depot arcs are significantly higher than for customer arcs regardless of the instance set. The UPPPVS instances even support the recommendation of Toth and Vigo (2003) that using all depot arcs in the generator arc list can be a sensible decision. This result also makes using different sparsification criteria and strengths for depot arcs an interesting avenue for future research (see, e.g., Schneider et al. 2017).

Arc length ratios: For each instance, we calculate an arc length ratio to analyze how much shorter BKS arcs are in relation to the average length of an arc in the complete instance graph. We differentiate between depot arcs (the average length of the depot arcs in the BKS is related to the average length of the depot arcs in the instance graph) and customer arcs (analogously). Table 2 shows the average values computed for the depot arcs (r_d) and the customer arcs (r_c) of the two instance sets. The results show

	IFG	UPPPVS
r_d	0.32	0.78
r_c	0.07	0.11

Table 2: Average arc length ratio of depot (r_d) and customer arcs (r_c).

clear differences between the two instance sets: while the arc length ratios of the customer arcs are

only slightly smaller for the IFG instances, the depot arc length ratios for the IFG instances are much smaller than those of UPPPVS. This confirms the need of a weaker sparsification for the UPPPVS and substantiates the need of intelligent sparsification techniques for treating depot arcs.

Tightness of capacity constraints: For each instance, we calculate the average capacity utilization in the BKS, defined as the ratio of the total customer demand and the vehicle capacity multiplied with the number of routes. Averaging over all instances, we obtain a capacity utilization of 91.4% for the IFG and 97.4% for the UPPPVS instances, indicating that the capacity constraints are more binding for the UPPPVS instances. As a result, to obtain optimal solutions, the included arcs have to enable a high capacity utilization and might therefore be longer. This observation contributes to explain why weaker sparsification is beneficial for the UPPPVS instances.

Vertex distribution: Finally, the distribution of the vertices in the Euclidean plane provides additional explanation. While customers are always located uniformly in the IFG instances, they are placed in different fashions, some of them containing customer clusters, in the UPPPVS instances. If these clusters have strongly differing distances from the depot, it can easily happen that all depot arcs in the list of generator arcs connect the depot with customers from only one or at least a limited proportion of the clusters. This leads to a reduced flexibility of the search to investigate different possibilities of connecting the depot and the customer clusters. A weaker sparsification mitigates this effect.

Summarizing, for the IFG benchmark set, the simple sparsification criterion arc length seems suitable to classify good arcs. As a consequence, short generator arc lists, i.e., strong sparsification, are sufficient to generate good solutions, something desirable for GLS. For the UPPPVS benchmark, on the other hand, arc length alone is not a good criterion so that much longer generator arc lists, i.e., weaker sparsification, are necessary to identify good solutions. Likewise, more sophisticated criteria are needed for the UPPPVS instances to distinguish good from bad arcs, which could allow a strong sparsification to perform well.

Thus, there are two main takeaways from our results regarding the granularization. First, if one has a very good idea about what characterizes promising arcs, a strong sparsification can be used. If not, a weaker sparsification might be necessary to obtain high-quality solutions. Second, good choices for the remaining design decisions seem to be independent of the sparsification strength.

Combinations of design decisions Table 3 lists the combinations of the four design decisions that constitute the non-dominated variants. Apparently, there seems to be no pattern in the sense that one design decision, say composite neighborhood definition, exclusively goes with another one, say sequential improvement for the pivoting rule. We see, however, that all three Pareto variants of the UPPPVS set are also among the non-dominated variants for the IFG instances when changing the sparsification strength (variants 3 and 7, 4 and 8, 5 and 9 match). This supports our point that the three decisions on neighborhood operators, pivoting rule, and search neighborhood are mainly independent of the sparsification strength. Even more, good individual design decisions are not tied to a specific combination with others, but they seem to work well when combined with other good decisions.

Variant	Neighborhood operators	Pivoting rule	Search neighborhood	Sparsification strength
IFG				
1	2/2*/rel/ex/strel	FI	VND	strong
2	2/2*/rel/ex/strel	3-SI	composite	strong
3	all	FI	composite	strong
4	all	FI	VND	strong
5	all	3-SI	VND	strong
6	all	R-10-FI	VND	strong
UPPPVS				
7	all	FI	composite	medium
8	all	FI	VND	weak
9	all	3-SI	VND	weak

Table 3: Non-dominated combinations of design decisions.

5 Conclusion

In this paper, we investigate the effect of different choices concerning the set of neighborhood operators, the sparsification strength, the pivoting rule, and the definition of the complete search neighborhood in a granular search for the CVRP. Based on extensive numerical studies on two CVRP benchmark sets and the use of a Wilcoxon signed-rank test to find non-dominated algorithmic variants, we make the following observations:

- Using a large number of neighborhood operators results in superior algorithmic variants.
- When using a restricted number of neighborhood operators, including relocate and/or string-relocate is beneficial.
- Defining the search neighborhood in VND fashion compared to a composite neighborhood is the more frequent choice in the non-dominated algorithmic variants.
- As a pivoting rule, first improvement (and its randomized version) and sequential improvement appear superior to best improvement.
- The accurate strength of sparsification strongly depends on the sparsification criterion used and the characteristics of the instances to solve.
- An analysis of the minimum sparsification factors, the arc length ratios, the tightness of capacity constraints, and the distribution of the vertices in the plane can help in the design of suitable granularization techniques.

Future research will investigate whether the gained insights can be used to design a state-of-the-art granular tabu search for the CVRP. This would provide an important contribution along the lines of developing successful heuristics which remain conceptually simple. Here, an important question is how to treat depot arcs when sparsifying the generator arc set.

While our computational analysis is limited to the CVRP as use case, we expect our findings to transfer to a large part also to other VRP variants. Future research may include similar analyses on such variants and the investigation of the given design recommendations when developing local-search-based algorithms for VRP variants.

Acknowledgments This research was funded by the Deutsche Forschungsgemeinschaft (DFG) under grants no. IR 122/7-1 and SCHN 1497/1-1. Simulations were performed with computing resources granted by RWTH Aachen University under project rwth0335.

References

F. Arnold and K. Sörensen. Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research*, 105:32–46, 2019.

O. Beek, B. Raa, W. Dullaert, and D. Vigo. An efficient implementation of a static move descriptor-based local search heuristic. *Computers & Operations Research*, 94:1–10, 2018.

M. Coffin and M. J. Saltzman. Statistical analysis of computational tests of algorithms and heuristics. *INFORMS Journal on Computing*, 12(1):24–44, 2000.

L. Costa, C. Contardo, and G. Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985, 2019.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

J. W. Escobar, R. Linfati, M. G. Baldoquin, and P. Toth. A granular variable tabu neighborhood search for the capacitated location-routing problem. *Transportation Research Part B: Methodological*, 67:344–356, 2014a.

J. W. Escobar, R. Linfati, P. Toth, and M. G. Baldoquin. A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem. *Journal of Heuristics*, 20(5):483–509, 2014b.

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

J. B. Gauthier and S. Irnich. Inter-depot moves and dynamic-radius search for multi-depot vehicle routing problems. Technical Report LM-2020-03, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany, 2020.

N. Gould and J. Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software*, 43(2), 2016.

P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5): 802–817, 2006.

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

G. Kindervater and M. Savelsbergh. Vehicle routing: Handling edge exchanges. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 10, pages 337–360. Wiley, 1997.

N. Labadie, R. Mansini, J. Melechovský, and R. Wolfler Calvo. The team orienteering problem with time windows: An LP-based granular variable neighborhood search. *European Journal of Operational Research*, 220(1):15–27, 2012.

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

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

Z. Lü, J.-K. Hao, and F. Glover. Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics*, 17(2):97–118, 2011.

A. Mjirda, R. Todosijević, S. Hanafi, P. Hansen, and N. Mladenović. Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem. *International Transactions in Operational Research*, 24 (3):615–633, 2017.

R. Moretti Branchini, V. Amaral Armentano, and A. Løkketangen. Adaptive granular local search heuristic for a dynamic vehicle routing problem. *Computers & Operations Research*, 36(11):2955–2968, 2009.

M. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47(1):75–85, 1990.

M. Schneider and M. Löffler. Large composite neighborhoods for the capacitated location-routing problem. *Transportation Science*, 53(1):301–318, 2019.

M. Schneider, F. Schwahn, and D. Vigo. Designing granular solution methods for routing problems with time windows. *European Journal of Operational Research*, 263(2):493–509, 2017.

P. Toth and D. Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.

P. Toth and D. Vigo, editors. *Vehicle routing: Problems, methods, and applications*, volume 18 of *MOS-SIAM Series on Optimization*. SIAM, 2nd edition, 2014.

E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

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

Appendix

Sets of neighborhood operators

The following 17 sets of neighborhood operators are considered in our computational study:

- 2/2*/rel/ex
- 2*/rel/ex/strel
- 2*/rel/ex/stex
- 2*/rel/ex/stexi
- 2/rel/ex/strel
- 2/rel/ex/stex
- 2/rel/ex/stexi
- 2/2*/ex/strel
- 2/2*/ex/stex
- 2/2*/ex/stexi
- 2/2*/rel/strel
- 2/2*/rel/stex
- 2/2*/rel/stexi
- 2/2*/rel/ex/strel
- 2/2*/rel/ex/stex
- 2/2*/rel/ex/stexi
- all (2/2*/rel/ex/strel/stex/stexi)