Hybridizing Large Neighborhood Search and Exacts Methods for Generalized Vehicles Routing Problems with Time Windows

Dorian Dumez, Christian Tilk, Stefan Irnich, Fabien Lehuédé, Olivier Péton

Hybridizing Large Neighborhood Search and Exacts Methods for Generalized Vehicles Routing Problems with Time Windows

Dorian Dumez[2], Christian Tilk*[1], Stefan Irnich[1], Fabien Lehuédé[2], and Olivier Péton[2]

[1]Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, 55099 Mainz, Germany.
[2]IMT Atlantique, Laboratoire des Sciences du Numérique de Nantes (LS2N, UMR CNRS 6004), Nantes, France

September 9, 2020

Abstract

Delivery options are at the heart of the generalized vehicle routing problem with time windows (GVRPTW) allowing that customer requests are shipped to alternative delivery locations which can also have different time windows. Recently, the vehicle routing problem with delivery options was introduced into the scientific literature. It extends the GVRPTW by capacities of shared locations and by specifying service-level constraints defined by the customers' preferences for delivery options. The vehicle routing problem with delivery options also generalizes the vehicle routing problem with home roaming delivery locations and the vehicle routing problem with multiple time windows. For all these GVRPTW variants, we present a widely applicable matheuristic that relies on a large neighborhood search (LNS) employing several problem-tailored destruction operators. Most of the time, the LNS performs relatively small and fast moves, but when the solution has not been improved for many iterations, a larger destruction move is applied to arrive in a different region of the search space. Moreover, an adaptive layer of the LNS embeds two exact components: First, a set-partitioning formulation is used to combine previously found routes to new solutions. Second, the Balas-Simonetti neighborhood is adapted to further improve already good solutions. These new components are in the focus of our work and we perform an exhaustive computational study to evaluate four configurations of the new matheuristic on several benchmark instances of the above-mentioned variants. On all the benchmark sets, our matheuristic is competitive with the previous state-of-the-art methods. Without manual problem-specific re-configurations of the matheuristic, we provide 81 new best-known solutions.

Keywords: vehicle routing; delivery options; time windows; matheuristic; large neighborhood search

1 Introduction

Delivery options are at the heart of the well-known *generalized vehicle routing problem with time windows* (GVRPTW, Moccia *et al.*, 2012). Delivery options allow customer requests to

*Corresponding author.

be shipped to alternative delivery locations which can also have different time windows. In this paper, we present and computationally evaluate a new matheuristic that can deal with several generalizations of the GVRPTW.

The most general of these is the *vehicle routing problem with delivery options* (VRPDO). The VRPDO has been first described in (Dumez *et al.*, 2020; Tilk *et al.*, 2020) and captures one of the recent trends in last-mile package delivery related to the introduction of delivery options. (Tilk *et al.*, 2020) provide a more detailed introduction to the VRPDO in the context of parcel delivery and last-mile vehicle routing problems (Cattaruzza *et al.*, 2017). In last-mile package delivery, a delivery option can be, e.g., the delivery to an individual home location, to the customer's car trunk when parking positions are known and communicated in advance, or to shared delivery locations such as apartment buildings, shops, and lockers. Motivated by the real-world characteristics of some delivery options, the VRPDO further extends the GVRPTW by service-level constraints defined by the customers' preferences for options and by location capacities that must be respected. For example, a customer may prefer home delivery over delivery to a nearby locker, and locker delivery over delivery to a nearby shop (preferences can depend on the time of the day). Moreover, lockers and also smaller shops certainly have only limited capacity to store parcels until they are finally picked up by customers. Such location-specific capacities are, therefore, taken into account in the VRPDO.

The VRPDO also generalizes other problems such as the *vehicle routing problem with roaming delivery locations* (VRPRDL, Reyes *et al.*, 2017) in which options result from deliveries to a customer's car trunk, and the *vehicle routing problem with multiple time windows* (VRPMTW, Favaretto *et al.*, 2007)) in which all options of a customer refer to the same physical location but have disjoint time windows.

The contribution of this work is the development of a powerful matheuristic that is sufficiently general to cope with all *vehicle routing problem* (VRP) variants mentioned above. The new matheuristic relies on a *large neighborhood search* (LNS) employing several destruction operators. Most of the time, the LNS performs fast and small moves, but when the solution has not been improved for many iterations, a larger destruction move is applied to arrive in a different region of the search space. The idea of mixing small and large moves within an LNS was introduced in the paper by Dumez *et al.* (2020) where specific neighborhoods were introduced for the VRPDO. In the paper at hand, we improve the results of Dumez *et al.* (2020) by a better control of large and small destructions. Moreover, the focus of our research is *not* on designing an LNS, but on an adaptive layer of the matheuristic that embeds two exact components: First, a *set-partitioning problem* (SPP) is solved from time to time to combine previously found routes into new solutions. Second, the *Balas-Simonetti* (BS, Balas and Simonetti, 2001) neighborhood is adapted to further improve already good solutions. The BS neighborhood is an exponentially-sized neighborhood in which a best improving solution can be found by solving a shortest-path problem in a layered graph. The two components are optional, giving rise to four different *configurations* of the matheuristic.

The experimentation with the two exact components complementing the LNS follows the general trend towards hybridization and matheuristics for difficult VRP variants. For example, Moccia *et al.* (2012) embed a dynamic-programming component in a tabu search for the GVRPTW, Tellez *et al.* (2018) solve a set-partitioning problem in an LNS for a rich dial-a-ride problem, and Toffolo *et al.* (2019) use the BS neighborhood in a structural decomposition approach for the *capacitated vehicle routing problem*. Our idea behind the two exact components in the LNS is that delivery options require a 'global view' on the search space, because the usefulness of an option is observable only when it is combined with suitable other options. Thus, simple local modifications of a solution alone will typically not show the usefulness of an option. What is required is larger modifications that result from several simultaneous changes in the

assignment of customer requests to routes, the selection of possible options, and the routing, i.e., sequence in which deliveries are performed. LNS by itself and also the two exact components pursue this ideas.

In an exhaustive computational study, we evaluate the four configurations of our matheuristic on several benchmark instances of the GVRPTW, VRPRDL, VRPMTW, and VRPDO. On all benchmark sets, at least one configuration of the matheuristic, often several, or even all configurations are competitive with the previous state-of-the-art methods. Without manual variant-specific parameter tuning of the matheuristic, we provide 81 new best-known solutions.

The remainder of this paper is organized as follows. Section 2 formally defines the GVRPTW variants. We describe the matheuristic with small and fast LNS moves and the two exact components, i.e., the SPP formulation and the BS neighborhood and its adaption to the GVRPTW variants, in Section 3. Section 4 presents the computational experiments and their results. Final conclusions are drawn in Section 5.

2 Problem Variants

In this section, we formally introduce the considered problem variants. We start with the VRPDO, because it is the most general variant. Afterwards, GVRPTW, VRPRDL and VRPMTW are briefly described and it is explained how they can be modeled as special cases of the VRPDO.

2.1 Vehicle Routing Problem with Delivery Options

The VRPDO is the problem of selecting delivery options, exactly one for each customer, and determining a cost-minimal set of feasible routes that serve the selected delivery options while respecting location-capacity and service-level constraints.

Let N be the set of customers (=delivery requests), L be the set of locations, and $P = \{1, 2, \dots, \bar{p}\}$ be the priority levels. A delivery option is a triple composed of a customer, location, and priority level. Formally, let $O \subset N \times L \times P$ be the set of delivery options. For an option $o \in O$, we write $n_o \in N$ for its customer/delivery request, $\ell_o \in L$ for its location, and $p_o \in P$ for its priority level. The priority indicates how much the customer prefers this option, smaller numbers indicate higher customer satisfaction. Additionally, each option o has a service time s_o.

A request $n \in N$ is characterized by a demand q_n, e.g., given by the number of parcels to deliver to that customer. The request can be served by choosing one of the options $O_n^N = \{(n_o, \ell_o, p_o) \in O : n_o = n\}$.

With a location we model all activities that can take place at the same physical place. Accordingly, we define $O_\ell^L = \{(n_o, \ell_o, p_o) \in O : \ell_o = \ell\}$ as the set of options belonging to location $\ell \in L$. An *individual* delivery location ℓ is one with a unique option, i.e., $|O_\ell^L| = 1$. Otherwise, we denote a location as a *shared* delivery location. Let $L^m = \{\ell \in L : |O_\ell^L| > 1\}$ be the set of shared delivery locations. Shared locations $\ell \in L^m$ have a limited capacity C_ℓ in terms of the number of shipments that can be delivered there. Moreover, we assume that at all locations $\ell \in L$ have an associated time window $[a_\ell, b_\ell]$ that describes the time period in which deliveries can be performed.

A fleet of K homogeneous vehicles with capacity Q is housed at the depot location $\ell_0 \in L$. For each pair of locations ℓ and $\ell' \in L$, the travel time $t_{\ell\ell'}$ and the travel cost $c_{\ell\ell'}$ are given. The travel time $t_{\ell\ell'}$ also includes a preparation time, e.g., for parking a vehicle at ℓ' before the actual delivery at ℓ' can start.

Service-level constraints are modeled with numbers $\beta_p \in [0, 1]$ for $p \in P$. The value β_p is the minimum percentage of options of service level not greater than p that must be chosen. For example, $\beta_2 = 0.8$ means that at least 80 % of the chosen options must have service level 1 or 2.

The capacities of the shared delivery locations and required service levels add synchronization constraints to the VRPDO, defined by Drexl (2012) as '*at any point in time, the total utilization or consumption of a specified resource by all vehicles must be less than or equal to a specified limit*'. In the VRPDO, resource consumption is cumulative over the time horizon. Hence, they can also be interpreted as inter-tour resource constraints as defined in (Hempsch and Irnich, 2008).

A route $r = (0, o_1, \ldots, o_h, 0)$ is as sequence of options in which the artificial options $o_0 = 0$ and $o_{h+1} = 0$ represent the visit of the depot location ℓ_0 at the start and end of the route, respectively. The demand served by route r is $q(r) = \sum_{j=1}^{h} q_{n_{o_j}}$, so that r is capacity-feasible if $q(r) \leq Q$ holds. A route is time-window feasible if there exists a schedule $(T_0, T_1, \ldots, T_h, T_{h+1}) \in \mathbb{R}^{h+2}$ which complies with the option service times, travel times, and time windows, i.e., if $T_{j-1} + t_{\ell_{o_{j-1}}, \ell_{o_j}} + s_{o_{j-1}} \leq T_j$ for all $1 \leq j \leq h+1$ (assuming $s_{o_0} = 0$) and $[T_j, T_j + s_{o_j}] \subseteq [a_{\ell_{o_j}}, b_{\ell_{o_j}}]$ for all $0 \leq j \leq h+1$. A route r is feasible if it fulfils both capacity and time-window constraints. The cost of a route is the sum of the travel cost between the consecutively visited locations, i.e., $c_r = \sum_{j=1}^{h+1} c_{\ell_{o_{j-1}}, \ell_{o_j}}$.

A solution S to VRPDO is a set of feasible routes that selects exactly one option per customer. Let $O(S)$ be the set of options selected in the solution S. Then, the requirement that S selects exactly one option per request translates into $|O(S) \cap O_n^N| = 1$ for all $n \in N$. The solution fulfills the location-capacity constraints if $|O(S) \cap O_\ell^L| \leq C_\ell$ for all $\ell \in L^m$. It fulfills the service-level constraints if

$$|\{o \in O(S) : p_o \leq p\}| \geq \beta_p |N| \qquad \Leftrightarrow \qquad |\{o \in O(S) : p_o > p\}| \leq (1 - \beta_p)|N| \tag{1}$$

for all $p \in P$. (Note that it suffices to test these conditions for $p \in P \setminus \{\bar{p}\}$, because $\beta_{\bar{p}} = 1$ is inevitable.) A solution S is feasible if it fulfills both location-capacity and service-level constraints. The objective of the VRPDO is to first minimize the number of vehicles, and second to minimize the routing cost $\sum_{r \in S} c_r$ over all feasible solutions.

Summarizing, the VRPDO is a GVRPTW with additional resources synchronization constraints (Drexl, 2012). Up to now, there are only two papers addressing the VRPDO: Tilk *et al.* (2020) solve the VRPDO exactly with a branch-price-and-cut algorithm and report provably optimal solutions for instances with up to 50 customers and 100 options. Dumez *et al.* (2020) use an LNS matheuristic to solve the VRPDO heuristically and report solutions on instances with up to 200 customers and 400 options.

2.2 Generalized Vehicle Routing Problem with Time Windows

The GVRPTW (Moccia *et al.*, 2012) is a direct generalization of the *vehicle routing problem with time windows* (VRPTW, Savelsbergh, 1985; Desaulniers *et al.*, 2014), in which customers has to be served at one of their alternative delivery locations respecting the corresponding time window. Each alternative delivery location defines a delivery option for the customer. Thus, the GVRPTW can be modeled and solved as a VRPDO without synchronized resources. To the best of our knowledge, (Moccia *et al.*, 2012) is the only article that deals with the heuristic solution of the GVRPTW. They propose an incremental tabu search using a dynamic-programming component that allows changing customers' locations when inserting a customer in a route. The tabu search provides solutions for instances with up to 120 customers in a few hundred seconds.

2.3 Vehicle Routing Problem with (Home and) Roaming Delivery Locations

The VRPRDL, introduced by Reyes *et al.* (2017), specifically models the delivery to the trunk of cars. Customers must a priori specify where they are over the planning horizon, thereby defining

different delivery options. The VRPRDL can be seen as a special case of the GVRPTW with non-overlapping time windows for the delivery options of each customer (Ozbaygin *et al.*, 2017).

The VRPHRDL is an extension of the VRPRDL with an additional so-called *home option* for each customer (Ozbaygin *et al.*, 2017). This additional delivery option has a non-constraining time window, e.g., identical to the planning horizon. Both problems are special cases of the GVRPTW and can, therefore, be modeled and solved as VRPDO without synchronized resources.

Reyes *et al.* (2017) propose a variable neighborhood search to solve the VRPRDL. It embeds a dynamic-programming algorithm to optimize the travel distance of a given customer sequence. Ozbaygin *et al.* (2017) develop a branch-and-price algorithm to solve the VRPRDL and VRPHRDL with up to 120 customers. Lombard *et al.* (2018) solve smaller instances of a stochastic VRPRDL.

2.4 Vehicle Routing Problem with Multiple Time Windows

The VRPMTW was introduced by Favaretto *et al.* (2007) as an extension of the VRPTW where each customer can have multiple time windows. If one considers each time window as a different option, the VRPMTW is a special case of the VRPDO. All delivery locations of a customer are at the same physical place. Options of two customers, however, always refer to different physical places.

The objective of the VRPMTW is either minimizing the travel distance or travel duration (total travel, service, and waiting time). In both cases, a fixed cost per route is included in the objective function fostering that the number of employed vehicles is kept small.

Belhaiza *et al.* (2014) propose a hybrid variable neighborhood tabu search and a set of benchmark instances with 100 customers. A revised version of the approach was described in (Belhaiza *et al.*, 2017) as a hybrid genetic variable neighborhood search. Larsen and Pacino (2019) solve the VRPMTW with an adaptive LNS with a problem-tailored insertion procedure. They generalize the forward time slack procedure of Savelsbergh (1985) to take into account all the time windows of the visited customers. Thus, the time windows used to visit the customers of a route can be changed to insert a new customer in this route. Hoogeboom *et al.* (2020) describe an adaptive variable neighborhood search relying on a generalization of the forward time slacks for the VRPMTW with duration-minimization objective.

3 LNS-based Matheuristic with Exact Components

In an LNS algorithm, the current solution is iteratively improved by removing a part of it (a.k.a. destroy, ruin) and reinserting the removed parts (a.k.a. repair, recreate). This process repeats until a stopping criterion is met, e.g., an iteration or time limit. LNS was first introduced by Shaw (1998) in a constraint programming context. The potential of solving a broad variety of VRPs with LNS was emphasized by Ropke and Pisinger (2006b,a); Pisinger and Ropke (2007). They proposed an adaptive version of LNS, known as ALNS, consisting of multiple destroy and repair operators adaptively selected according to their past performance. As surveyed in (Pisinger and Ropke, 2019), LNS has been successfully applied to many variants of the VRP.

As it is clearly beyond the scope of the paper at hand to survey LNS, we highlight two leading sources of inspiration for our matheuristic. First, Christiaens and Vanden Berghe (2019) developed a fast LNS based on small removals and fast greedy insertion heuristics as repair operators. Thanks to these two factors, their LNS can perform a very high number of iterations, which somehow compensates the lack of a local search in LNS. The LNS of Christiaens and

Vanden Berghe proved competitive with state-of-the-art algorithms on many VRPs, including the fundamental and intensively-studied VRPTW.

Preliminary experiments on GVRPTW variants indicate that fast and small moves alone are not sufficient for finding excellent solutions for the VRP variants considered here. Additionally, some more global modifications are needed from time to time. Therefore, we follow an idea of Hemmelmayr *et al.* (2012) who developed the following ALNS for the 2-echelon VRP: Mostly, their ALNS applies operators that modify only the second echelon. However, if the current solution is not improved for several iterations, a specific operator modifies the first echelon. Because of the structure of the 2-echelon VRP, such a first-echelon move strongly affects the resulting preconditions for the second echelon.

Both strategies can be transmitted to the GVRPTW case: the matheuristic mostly performs fast and small local moves and, when the solution has not been improved for some iterations, a larger destroy followed by one of the standard repair operators is applied.

In what follows, we describe the specific exact components of the new matheuristic relying on a mixed integer programming (MIP) solver and dynamic programming, respectively. The overall outline is first presented in Section 3.1. The Balas-Simonetti neighborhood and its adaption to the GVRPTW variants are described in Section 3.2. Finally, the set-partitioning formulation for all GVRPTW variants and the adaptive layer are detailed in Section 3.3.

3.1 Algorithm Outline

Recall that a solution has been defined as a set of feasible routes (see Section 2), where each route is represented as a sequence of options. Also in our matheuristic (`MathHeu`), a solution S comprises only feasible routes (w.r.t. vehicle-capacity and time-window constraints). Moreover, all options of all routes must refer to different customers, i.e., a packing solution. In addition, all feasible solutions fulfill the service-level requirements and the capacity constraints of the shared locations. However, as proposed by Pisinger and Ropke (2007), we allow *partial solutions*, i.e., solutions that do not serve all the customers in the course of the algorithm. As a consequence, we define the *modified cost* of a solution S as

$$f'(S) = f(S)\left(1 + \zeta\frac{|B(S)|}{|N|}\right)$$

where $B(S)$ is the set of customer requests that are not served by solution S (often called the request bank). In our experiments, we use a static value $\zeta = 20$.

As the LNS for the VRPDO developed in Dumez *et al.* (2020), our `MathHeu` is composed of two local and seven global destroy operators as well as six repair operators, all adapted from the literature. The operators are summarized in Table 1. Following the ideas of Christiaens and Vanden Berghe (2019), there is a probability of 0.3 to not remove a customer that should have been removed by the destroy operator. We called it *the blink at the deletion*. For all the operators, we are using their adaption to the GVRPTW variants as described in Dumez *et al.* (2020).

Table 1: Destroy and Repair Operators

Type	Operator	Source
local destroy	distance related customer removal	(Shaw, 1998)
	split string removal	(Christiaens and Vanden Berghe, 2016)
destroy	random option removal	(Ropke and Pisinger, 2006a)
	time related customer removal	(Ropke and Pisinger, 2006a)
	cluster removal	(Pisinger and Ropke, 2007)
	route removal	(Nagata and Bräysy, 2009)
	zone removal	(Demir *et al.*, 2012)
	historical knowledge node removal	(Demir *et al.*, 2012)
	SDL oriented random removal	(Dumez *et al.*, 2020)
repair	2-regret	(Ropke and Pisinger, 2006a)
	ejection search	(Nagata and Bräysy, 2009)
	random order best insertion	(Christiaens and Vanden Berghe, 2016)
	largest first best insertion	(Christiaens and Vanden Berghe, 2016)
	preferred best insertion	(Dumez *et al.*, 2020)
	SDL-regret	(Dumez *et al.*, 2020)

All six repair operators (see Table 1) rely on constant-time feasibility checks when inserting a customer into a route. To this end, the utilization of each shared location and the current priority fulfillment of each priority level is seamlessly recorded. In addition, the load onboard of each vehicle is recorded and forward time slacks (Savelsbergh, 1992) are computed at each option in a route. As in Dumez *et al.* (2020), the insertion of a request n between two visited options in a route is tested considering all its possible options O_n^N.

Note that the focus of the paper at hand is not to further fine-tune the LNS for all GVRPTW components. Instead, the primary focus is on defining and analyzing the impact that the new adaptive layer and the two exact components have on the performance, i.e., solution quality and speed.

We can now present a synopsis of the new `MathHeu` for all GVRPTW variants in Algorithm 1. In this pseudo code, S is the current solution, S^* is the best-found solution, and S' is a copy of the current solution to be modified. Moreover, Σ^+ is the set of repair operators, Σ^- the set of destroy operators, and $\Sigma^-|_{\text{local}} \subset \Sigma^-$ is the set of local destroy operators. Each operator has a given and fixed probability to be selected (see Section 4.1). The variable *iter* counts the number of iterations since the last new best solution was found or the last large destroy was performed. The pool of routes to be used in the SPP is denoted by $\mathcal{P}$ (see Section 3.3).

The algorithm is initialized by setting *iter* to zero and the pool $\mathcal{P}$ of routes is cleared (Line 1). The main loop of the `MathHeu` is given by the Lines 3 to 20. In each iteration, either a local or large destruction operator is selected depending on the value of *iter* and the input parameter ω: If $iter < \omega$, the iteration counter is increased, the current solution is copied and a small destruction using a local destroy operator in $\sigma^- \in \Sigma^-|_{\text{local}}$ with destruction size in $[\delta_{\min}, \Delta_{\min}]$ is performed (Lines 4 to 5). Otherwise, the counter *iter* is reset, the best-found solution is copied and a large destruction is performed, i.e., the destroy operator is randomly selected in Σ^- and the destruction size is chosen at random in $[\delta_{\text{big}}, \Delta_{\text{big}}]$ (Lines 7 to 8). A repair operator is randomly selected in $\sigma^+ \in \Sigma^+$ (Line 9). Then, the combination of the selected operators, σ^- and σ^+, is applied to solution S' in Line 10.

If the resulting solution serves all the customers and its cost is less than ϵ percent away from the cost of the best-found solution, the algorithm tries to improve S' with the Balas-Simonetti component at Line 12 (see Section 3.2 for details). In Line 13, the routes of the newly generated solution are stored into the pool of routes. The new solution becomes the current solution in

Algorithm 1: LNS-based Matheuristic (`MathHeu`) with two Exact Components

input : operators $\Sigma^+, \Sigma^-, \Sigma^-|_{\text{local}}$, parameters $[\delta_{\min}, \Delta_{\min}], [\delta_{\text{big}}, \Delta_{\text{big}}], \omega$, initial solution S
output: best-found solution S^*

1 $\mathcal{P} \leftarrow \emptyset$, *iter* $\leftarrow 0$
2 **while** *the time budget is not reached* **do**
3 **if** *iter* $< \omega$ **then**
4 *iter* $\leftarrow$ *iter* $+ 1$, $S' \leftarrow S$
5 randomly select a destroy operator $\sigma^- \in \Sigma^-|_{\text{local}}$, a destruction size $\Phi \in [\delta_{\min}, \Delta_{\min}]$
6 **else**
7 *iter* $\leftarrow 0$, $S' \leftarrow S^*$
8 randomly select a destroy operator $\sigma^- \in \Sigma^-$, a destruction size $\Phi \in [\delta_{\text{big}}, \Delta_{\text{big}}]$
9 randomly select a repair operator $\sigma^+ \in \Sigma^+$
10 $S' \leftarrow \sigma^+(\sigma^-(S', \Phi))$
11 **if** $f(S') < (1+\epsilon) f(S^*)$ ***and*** S' *is feasible* **then**
12 improve S' with the Balas-Simonetti neighborhood
13 add routes of S' to pool $\mathcal{P}$
14 **if** $f'(S') < f'(S)$ ***or*** *iter* $= \omega$ **then**
15 $S \leftarrow S'$;
16 **if** $f(S') < f(S^*)$ ***and*** S' *is feasible* **then**
17 $S^* \leftarrow S'$, *iter* $\leftarrow 0$
18 **if** *a sufficient number of routes is generated* **then**
19 improve S with the SPP and route set $\mathcal{P}$
20 $\mathcal{P} \leftarrow \emptyset$

Line 14 if its modified cost is smaller than the current or a large destruction was performed. In Line 16, the best-found solution may be updated and the counter *iter* is reset accordingly. Finally, when the pool of routes is big enough, the SPP is solved in Line 19 (details are provided in Section 3.3) and the pool is cleared afterwards.

In the following, we evaluate four configurations of the `MathHeu`:

`MH`:	without any exact component	(Lines 11–12 and 18–20 deactivated)
`MH+BS`:	with only the BS component activated	(Lines 18–20 deactivated)
`MH+SPP`:	with only the SPP component activated	(Lines 11–12 deactivated)
`MH+SPP+BS`:	with both exact components activated	

3.2 Balas-Simonetti Component

Balas (1999) proposed and analyzed a family of large-scale neighborhoods for the *asymmetric traveling salesman problem* (ATSP) that can be searched efficiently (in linear time in the size of the Hamiltonian path). The neighborhoods $\mathcal{N}^k_{BS}$ are parameterized by an integer $k \geq 2$ and Balas and Simonetti (2001) used them within a local-search algorithm for the ATSP and the ATSP with time windows.

In order to be self-contained, we very briefly summarize the Balas-Simonetti neighborhoods for the ATSP: Given an TSP Hamiltonian path $x = (x_0, x_1, \ldots, x_n, x_{n+1})$ the neighborhood $\mathcal{N}^k_{BS}(x)$, for a given value $k \geq 2$, consists of all tours $x' = (x_0, x_{\pi(1)}, \ldots, x_{\pi(n)}, x_{n+1})$, where π is a permutation of $\{1, \ldots, n\}$ that fulfills the following conditions: For any two indices $i, j \in \{1, \ldots, n\}$ with $i + k \leq j$, the inequality $\pi(i) < \pi(j)$ holds. It means that if a vertex x_i precedes a vertex x_j by at least k positions in the given path x, then x_i must also precede x_j in the neighbor x'. For a given value of the parameter k, a best neighbor $x' \in \mathcal{N}^k_{BS}(x)$ can be

determined in $O(k^2 2^{k-2}\, n)$ by solving a shortest-path problem in an auxiliary network (Balas, 1999), i.e., for a fixed value of the parameter k the neighborhood exploration is linear in n.

An example of an auxiliary network is depicted in Figure 1 for $k = 3$ and $n = 5$. The graph consist of stages/levels (depicted from left to right in the figure) that correspond to the positions in the Hamiltonian path. The vertices at each stage can be ordered into rows that are associated with an offset value α. In the following, the vertices of the auxiliary network are called *states*. Thus, each state in the auxiliary network is associated with the vertex $x_{j+\alpha}$ where j is the stage of the state and α the value associated with the state. Arcs exclusively go from states of a stage j to states of the subsequent stage $j + 1$. The general structure of the auxiliary network is described in several works, e.g., (Balas and Simonetti, 2001) and several subsequent articles (Irnich, 2008b; Tilk and Irnich, 2016; Hintsch and Irnich, 2018).

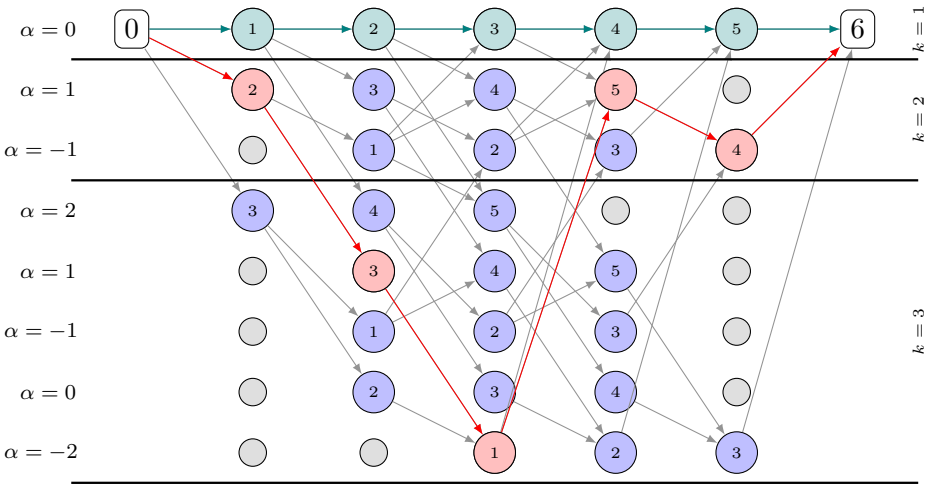

Figure 1: Example of the auxiliary network for $k = 3$ and $n = 5$

Every source-sink path, from 0 to $n+1$, in the auxiliary network corresponds to a neighbor x'. For example, the green sequence of states at the top row in Figure 1 corresponds to $x' = x = (0, 1, 2, 3, 4, 5, 6)$. The red sequence of states corresponds to the neighbor $x' = (0, 2, 3, 1, 5, 4, 6)$, i.e., a Hamiltonian path that respects the precedence constraint with respect to the initial tour x and the given parameter $k = 3$.

The structure of the auxiliary network, i.e., states and connecting arcs, depends only on k and n, but not on the given path x. The only difference between two auxiliary networks is the cost of the arcs that must be set to the distance between the considered customers. Consequently, if the BS neighborhood must be explored multiple times, the auxiliary network can be kept. Moreover, the auxiliary network for k' is always a state-induced subgraph of the auxiliary network for any larger $k > k'$ (indicated by the separating lines in Figure 1).

Adaptions. Next, we explain how to adapt the Balas-Simonetti neighborhood such that it can be used for all GVRPTW variants, including the VRPDO with the inter-route synchronization constraints imposed by location capacities and service-level requirements.

First, we generalize the auxiliary network such that it can deal with options. Contrary to the TSP, a delivery request n can in our applications be served using the different delivery options O_n^N. Hence, *meta states* encompass sets of states that represent all options of the respective customer as shown in Figure 2. All states for options of a customer are linked to all states for options of the subsequent customer.

Second, a *shortest-path problem with resources constraints* (SPPRC, Irnich and Desaulniers,

Figure 2: A sequence of customers in the auxiliary network for the ATSP and its generalization to delivery options in the GVRPTW. There are three, two and, four options per delivery request in this example, i.e., $|O_1^N| = 3$, $|O_2^N| = 2$, and $|O_3^N| = 4$.

2005) must be solved on the generalized auxiliary network, in contrast to solving one without resource constraints for the ATSP.

We solve the corresponding SPPRCs with a label-setting algorithm as follows: A label $\mathcal{L}_i = (i, C_i, T_i, Q_i, (R_i^p)_{p\in P}, (H_i^\ell)_{\ell\in L^m})$ represents a partial path from the depot 0 to a state i in the auxiliary network. Recall that a state i represents an option and therefore a delivery request n_i, a location ℓ_i, a priority level p_i, and a service time s_i are associated. The components of the label $\mathcal{L}_i$ are

(i) last state i of the partial path,
(ii) the accumulated routing cost C_i,
(iii) the earliest arrival time T_i at the location associated with i,
(iv) the accumulated demand (=load) Q_i in the vehicle,
(v) the number H_i^ℓ of shipments for each shared delivery location (for all $\ell \in L^m$), and
(vi) the number R_i^p of options o with priority p_o greater than p served along the partial path (for all $p \in P \setminus \{\bar{p}\}$).

The initial partial path in the auxiliary network starts from the depot 0 and the label $\mathcal{L}_0 = (0, 0, a_{\ell_0}, 0, \mathbf{0}, \mathbf{0})$. The label-setting algorithm extends labels along the arcs of the auxiliary network. Extending a label $\mathcal{L}_i = (i, C_i, T_i, Q_i, (R^p)_i^{p\in P\setminus\{\bar{p}\}}, (H_\ell)_i^{\ell\in L^m})$ along an arc (i, j) results in the label $\mathcal{L}_j = (j, C_j, T_j, Q_j, (R^p)_j^{p\in P\setminus\{\bar{p}\}}, (H_\ell)_j^{\ell\in L^m})$ defined by:

$$
\begin{aligned}
C_j &= C_i + c_{\ell_i\ell_j} \\
Q_j &= Q_i + q_{n_j} \\
T_j &= \max\{a_{\ell_j}, T_i + t_{\ell_i\ell_j} + s_i\} \\
H_j^\ell &= \begin{cases} H_i^\ell + 1, & \text{if } \ell_j = \ell \\ H_i^\ell, & \text{otherwise} \end{cases} && \text{for all } \ell \in L \\
R_j^p &= \begin{cases} R_i^p + 1, & \text{if } p_j > p \\ R_i^p, & \text{otherwise} \end{cases} && \text{for all } p \in P \setminus \{\bar{p}\}
\end{aligned}
$$

The extension is feasible, if the following constraints

$$
\begin{aligned}
Q_j &\le Q \\
T_j &\le b_j \\
H_j^\ell &\le C_\ell - \bar{H}^\ell, && \text{for all } \ell \in L^m \\
R_j^p &\le (1 - \beta_p)|N| - \bar{R}^p, && \text{for all } p \in P \setminus \{\bar{p}\}
\end{aligned}
$$

are fulfilled, where the values $\bar{H}^\ell$ for $\ell \in L$ and $\bar{R}^p$ for $p \in P \setminus \{\bar{p}\}$ are parameters that control the required slack in the location-capacity and service-level constraints. For the latter, please compare with (1) and (2d) in the next section.

We consider three different use cases for the BS component:

(1) the local optimization of a single route,
(2) the local optimization of the giant route, and
(3) the simultaneous local optimization of pairs of routes.
In case of a single route $r \in S$ taken from a current solution S, the parameters $\bar{H}^\ell$ and $\bar{R}^p$ for $p \in P \setminus \{\bar{p}\}$ can be set to the resource consumptions of all other routes $r' \in S \setminus \{r\}$.

In case of the giant route, all routes of a given solution S need to be joined as in (Irnich, 2008c) and (Hintsch and Irnich, 2018). The parameters $\bar{H}^\ell$ and $\bar{R}^p$ are set to 0. On arcs $(0, 0)$ joining two different routes, the resources T and Q are reset to a_{ℓ_0} and 0, respectively. In contrast, the resources H and R are kept at their current value enabling that the inter-route synchronization constraints are incorporated correctly.

In case of pairs (r_1, r_2) of routes, the two routes are joined leading to a partial giant route. The parameters $\bar{H}^\ell$ and $\bar{R}^p$ are then set to the resource consumptions of all other routes $r' \in S \setminus \{r_1, r_2\}$. Note that the BS neighborhood here also allows moving the depot vertices (in the middle) so that the route length of r_1 and r_2 can change.

Since all resource extensions are non-decreasing and resource consumptions are bounded from above, we can apply standard $\leq$-dominance (Irnich, 2008a). Obviously, resource H and R are obsolete in the GVRPTW, VRPRDL, VRPHRDL, and VRPMTW. However, due to a possibly large number of shared delivery locations and priorities in the VRPDO, the dominance relation can become rather weak in this variant.

We have conducted some preliminary experiments that have shown that the route-pairing version is the most effective (note that it includes the single route optimization). Solving the SPPRC for the giant route is excessively time-consuming, even on small instances and for small BS parameters $k \geq 2$. We therefore use the BS component only for all pairs of routes in the following.

Acceleration Techniques. As speed is essential, we devise six different acceleration techniques, two exact (they do not change the output of the labeling algorithm) and four heuristic acceleration methods (they may hinder the labeling algorithm to compute an optimal solution of the SPPRC).

On the exact side, we first use a bidirectional labeling algorithm (Righini and Salani, 2006). Note that backward extension and merging of labels follows standard rules (Irnich, 2008a).

Second, when pairing routes, we solve an SPPRC on every pair of routes in the solution, i.e., the routes will be used in several combinations. Therefore, some of the labels computed at the beginning of the forward and backward labeling are identical for pairs that share a common route and must be computed only once.

On the heuristic side, we first restrict the labeling to a limited discrepancy search (Feillet *et al.*, 2007). Therein, we limit the number of customers that can change their delivery option. The consequence is that all customer requests can still be permuted but only a limited number of the currently used options can be replaced by an alternative option of the respective request. In our experiments, we limit the number of customers that can change their delivery option to 3.

Second, we use a heuristic bounding strategy that discards labels if their cost is larger than some threshold value. The threshold value is computed as the difference between the best known cost and the sum of the best individual service cost of each non-served customer. For each solution computed in the course of the `MathHeu`, we compute a *customer-specific* service cost of each customer and update the best one if necessary: If a customer is served in an individual delivery location, its customer-specific service cost is the mean of the costs of the ingoing and outgoing arcs used to access it. If a customer is served in a shared delivery location,

the cost of the ingoing and outgoing arcs is divided by two times the number of customers served at this location by this route.

Third, a significant number of the resources of a label is dedicated to the capacity consumption of shared delivery locations which results in a relatively weak dominance. To strengthen the dominance, we ignore some of the shared delivery locations in the dominance test. To decide which shared delivery locations are ignored, the cost of the best solution that completely utilizes the capacity of a location is recorded in the LNS. We ignore those locations used only in solutions with a cost that is more than 10 % higher than the cost of the current best-found solution.

Fourth and finally, routes are sequences of options and multiple options taking place at the same shared delivery location can be exchanged without changing the cost of the solution. We break these symmetries by maintaining the order of the currently chosen options of the same shared delivery locations.

3.3 Set-Partitioning Component

The set-partitioning formulation for vehicle routing and scheduling problems has a long tradition (Foster and Ryan, 1976). It is nowadays widely used in column-generation approaches for many types of problems (Toth and Vigo, 2014). The extended model presented in the following was developed in (Dumez *et al.*, 2020) and (Tilk *et al.*, 2020).

Let R be a set of feasible routes, e.g., the pool $\mathcal{P}$ in Algorithm 1. For each route $r \in R$ and each option $o \in O$, the binary parameter α_r^o takes value 1 if the option $o \in O$ is served by the route $r \in R$, and 0 otherwise. The model uses binary variables λ_r for $r \in R$ that indicate if route r is used in the solution:

$$\min \sum_{r \in R} c_r \lambda_r \tag{2a}$$

$$\text{subject to} \sum_{r \in R} \sum_{o \in O_n^N} \alpha_r^o \lambda_r = 1 \qquad \forall n \in N \tag{2b}$$

$$\sum_{r \in R} \sum_{o \in O_\ell^L} \alpha_r^o \lambda_r \leqslant C^\ell \qquad \forall \ell \in L^m \tag{2c}$$

$$\sum_{r \in R} \sum_{o \in O : p_o > p} \alpha_r^o \lambda_r \leqslant (1 - \beta_p)|N| \qquad \forall p \in P \setminus \{\bar{p}\} \tag{2d}$$

$$\sum_{r \in R} \lambda_r \leqslant K \tag{2e}$$

$$\lambda_r \in \{0, 1\} \qquad \forall r \in R \tag{2f}$$

The objective (2a) minimizes the total cost of the solution, i.e., the sum of the cost of the routes used. The set-partitioning constraints (2b) state that each customer must be served exactly once. The capacity constraints for shared locations are given by (2c) and the service-level constraints by (2d). The fleet-size constraint (2e) sets the upper bound K on the number of routes used in the solution. Finally, the variable domains are given in (2f).

For solving the model with a MIP solver, it is known that typically set-covering formulations have shorter computation times than the respective set-partitioning formulations (see, e.g., Yıldırım and Çatay, 2015). If the travel times and costs fulfill the triangle inequality (always true for the benchmarks of Section 4), the partitioning constraints can be replaced by covering constraints, i.e., ≥ 1 instead of $= 1$ in (2b). In a set-covering solution, more than one option may be used to serve a customer. A simple greedy procedure can quickly repair and improve solutions

which overcover some customers. Note that the repaired solutions automatically respect the location-capacity and service-level constraints.

To further reduce the solution times, we extend the formulation by introducing binary variables y_o that indicate if option $o \in O$ is selected. The new y variables are coupled with the route variables via

$$\sum_{r \in R} \alpha_r^o \lambda_r = y_o \qquad \forall o \in O. \tag{2g}$$

We prioritize branching on these variables in the MIP solver.

4 Computational Experiments

In this section, we report the results of computational experiments that were conducted on GVRPTW, VRPRDL, VRPHRDL, VRPMTW, and VRPDO benchmarks with the four configurations of the matheuristic. The algorithms have been coded in C++ and compiled into 64-bit single-thread code with g++ 5.4.0. All experiments were performed using Linux, Ubuntu 16.04 LTS, running on an Intel Xeon X5650 @ 2.57 GHz. We use IBM Ilog CPLEX 12.8.0 (IBM, 2018) to solve the SPP and activate the options `branch_up_first` and `emphasis_on_hidden_feasible_solutions`.

Section 4.1 summarizes the parameter values used in the experiments. A comparison with algorithms from the literature on standard benchmark sets for the GVRPTW, VRPRDL, VRPHRDL, VRPMTW, and VRPDO is conducted in Sections 4.2 to 4.5, respectively. Finally, all results are discussed together and a summary is given in Section 4.6.

4.1 LNS Parameters and Adaptive Layer

We briefly sketch the parameter settings of the LNS, even though, our focus is primarily on the new matheuristic components. First, we do not dynamically adapt the probability of each operator as suggested by Ropke and Pisinger (2006b). Indeed, Turkeš *et al.* (2019) reviewed the LNS literature and conclude that this feature has, at best, a very little positive impact. Our preliminary experiments confirm this observation for the GVRPTW variants and our matheuristic configurations. Hence, the probability to choose an operator is constant and inversely proportional to its average run time as proposed in Dumez *et al.* (2020). Destroy operators are equiprobable. The two operators *random order best insertion* and *largest first best insertion* have a probability of 40 %, while all other repair operators have a probability of 5 % to be chosen. Moreover, we set the size of the small destruction interval to $[\delta_{\min}, \Delta_{\min}] = [0.01|N|, 0.1|N|]$ and the size of the large destruction interval to $[\delta_{\text{big}}, \Delta_{\text{big}}] = [0.1|N|, 0.3|N|]$. Our LNS always works in two phases: In the first phase, it reduces the number of routes, and in the second phase, it minimizes the routing cost using the established number of routes. The first phase is either stopped when half of the time budget has been used, or sooner when a feasible solution with the given number of routes has been found. The number of iterations between two big destructions is set to $\omega = 10|N|^{1.5}$ (we refer to the paper of Dumez *et al.*, 2020, for further details on the LNS configuration).

In many applications, the strength of LNS relies on its speed. Pisinger and Ropke (2007) report that their ALNS takes 146 seconds, on average, for performing 50,000 iterations on the Solomon instances for the VRPTW with 100 customers. With the same time budget, on the same instances, `MathHeu` performs 7.4 million iterations on average, i.e., approximately 150 times more iterations with a CPU that is only 1.91 times faster per thread according to (PassMark-Software, 2020). This is possible thanks to the small and fast destroy moves.

Next, we determine a strategy for the adaptive layer of `MathHeu`. The question is when and how to employ the exact components of Algorithm 1. We extend the procedure used in Tellez *et al.* (2018) to solve the fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity with an LNS coupled with SPP: They initially solve the SPP every 1 000 iterations, but, if the solver fails to prove optimality twice in a row the time between two calls is reduced by a quarter. We use the size $|\mathcal{P}|$ of the route pool instead of the consumed time to decide when the SPP component is invoked. More precisely, we solve formulation (2) with the MIP solver when the pool of routes R contains at least $\max(100, 38000 - 180|N|)$ different routes. This threshold size is increased by 60% if the solver has proven optimality twice in a row. Conversely, it is reduced by the same factor if the solver has failed to prove optimality twice in a row.

Both exact components are only applied if the cost of the best-known solution has improved by less than 1 % over the last 5ω iterations. Moreover, a good solution-quality-to-time compromise is a small value of $k = 5$. With this value the BS component is applied exclusively to solutions that serve all customers and with a cost smaller than 1.01 times of the cost of the best-known solution. The used parameter values come from preliminary experiments that were satisfactory.

The results presented in the following sections provide the following information:

Instance(s): Name of the benchmark instance (group)
#: Number of instances in the group
$|N|$: Number of customer requests
#veh: Upper bound computed/set on the number of vehicle/the fleet size
Σveh: Sum of the number of vehicles computed
Cost: Overall cost of the best computed solution
ΣCost: same, summed over the group of instances
#Best: Number of instances for which an algorithm has found a best-known solution

4.2 Results for the GVRPTW

We compare all four configurations of `MathHeu` with the *iterative tabu search* (`ITS`) of Moccia *et al.* (2012) on their benchmark containing instances with up to 120 customers. These instances are named `i-`n`-`$v_{\min}$`-`$v_{\max}$ where n is the number of customers, $v_{\min}$ the minimum, and $v_{\max}$ the maximum number of options per customer.

Moccia *et al.* (2012) consider the single objective of routing-cost minimization. For a fair comparison, we bound the number of available vehicles to the value computed in their work.

Moccia *et al.* (2012) performed their computational experiments on an Intel Core Duo @ 1.83 GHz and limited their algorithm to 10^5 iterations taking a total computation time of 8,105 seconds. In our matheuristic, we allocate a quota for the computation time for each instance according to the number of customers. In total, the `MathHeu` consume 524 seconds for all 20 benchmark instances. According to PassMark-Software (2020), on single thread benchmarks, their processor is 2.18 times slower than ours. It means that, on average, the `MathHeu` is configured to run approximately 7 times faster than the `ITS`.

The results are presented in Table 2. Note that it is not indicated in Moccia *et al.* (2012) whether their results are the best results out of several runs. For the four `MathHeu` configuration, the best solution out of five runs is taken into account. The best solution values are marked in bold.

All four `MathHeu` configurations clearly outperform the results of the `ITS`. The most BKS are found with `MH` and `MH+SPP+BS`. Regarding the routing cost, `MH+SPP` performs best and improves the results of ITS by 0.42 %. Moreover, `MH+SPP` provides an equivalent or better solution than `ITS` on 15 of the 20 instances. However, all four configurations of `MathHeu` perform rather

	ITS		Cost of MathHeu configurations			
Instance	#veh	Cost	MH	MH+BS	MH+SPP	MH+SPP+BS
i-030-04-08	4	3 498	**3 497**	**3 497**	**3 497**	**3 497**
i-030-08-12	4	2 866	**2 796**	**2 796**	**2 796**	**2 796**
i-040-04-08	6	**3 811**	**3 811**	**3 811**	**3 811**	**3 811**
i-040-08-12	6	**3 757**	3 768	3 768	3 768	3 768
i-050-04-08	8	5 447	5 447	5 447	**5 439**	5 447
i-050-08-12	8	**4 034**	**4 034**	**4 034**	**4 034**	**4 034**
i-060-04-08	8	5 919	**5 908**	5 926	5 919	5 926
i-060-08-12	8	**4 303**	**4 303**	**4 303**	**4 303**	4 332
i-070-04-08	10	**6 205**	6 246	6 246	6 224	6 246
i-070-08-12	10	4 645	**4 644**	**4 644**	**4 644**	**4 644**
i-080-04-08	12	7 425	**7 390**	7 394	7 394	7 394
i-080-08-12	12	5 734	5 686	**5 661**	5 692	**5 661**
i-090-04-08	11	**7 110**	7 187	7 182	7 187	7 215
i-090-08-12	11	5 810	5 903	5 869	5 830	**5 808**
i-100-04-08	14	7 455	7 308	7 349	**7 295**	**7 295**
i-100-08-12	14	6 703	**6 546**	**6 546**	6 585	6 606
i-110-04-08	16	8 719	8 696	8 720	8 711	**8 687**
i-110-08-12	16	6 281	**6 249**	6 487	6 338	6 310
i-120-04-08	15	8 512	**8 344**	8 357	**8 344**	**8 344**
i-120-08-12	15	6 833	6 829	6 829	**6 774**	**6 774**
Sum		115 069	114 592	114 866	**114 585**	114 595
#Best		6	**11**	8	10	**11**

Table 2: Results for the GVRPTW benchmark of Moccia *et al.* (2012).

similarly on the GVRPTW benchmark while the ITS perform worse. In total, we provide 14 new best-known solutions. Detailed instance-by-instance results can be found in Appendix A.

4.3 Results for the VRPRDL and VRPHRDL

We next compare the four MathHeu configurations with the exact branch-price-and-cut (BPC) algorithm of Tilk *et al.* (2020) on a benchmark set originally proposed by (Reyes *et al.*, 2017). As suggested by Ozbaygin *et al.* (2017), the instances should be modified such that the triangle inequality for travel cost and travel times holds. The modified benchmark that we also use consists of 120 randomly generated instances with a size ranging from 15 to 120 delivery requests, with a maximum of 6 options per request. The benchmark set is divided into 60 VRPRDL and 60 VRPHRDL instances.

Note that we cannot fairly compare with the variable neighborhood search of Reyes *et al.* (2017), because we only have access to the modified instances provided by Ozbaygin *et al.*. Since the BPC of Tilk *et al.* minimizes routing costs (as typically done in studies of exact algorithms), we only focus on routing-cost minimization. To this end, we bound the number of vehicles by the number of routes given in their solutions.

The time budget for the four configurations of the MathHeu is now set to 60 seconds for instances with up to 60 customers, and 300 seconds for the 120-customer instances. This is again a rather small computation time compared to the 2 hour and 6 hour time limit used in (Tilk *et al.*, 2020).

Aggregated results for the VRPRDL are shown in Table 3 and for the VRPHRDL in Table 4. Each line refers to a group of instances with identical number $|N|$ of customers. The BPC

			BPC		ΣCost of MathHeu configurations			
Instances	#	$\|N\|$	Σveh	ΣCost	MH	MH+BS	MH+SPP	MH+SPP+BS
1–5	5	15	24	**6 072**	**6 072**	**6 072**	**6 072**	**6 072**
6–10	5	20	27	**6 848**	**6 848**	**6 848**	**6 848**	**6 848**
11–20	10	30	68	**18 595**	**18 595**	**18 595**	**18 595**	**18 595**
21–30	10	60	129	**37 213**	**37 213**	**37 213**	**37 213**	**37 213**
31–40	10	120	189	**53 738**	53 759	53 876	**53 738**	53 761
41–50_v1	10	40	94	**29 838**	**29 838**	29 842	**29 838**	**29 838**
41–50_v2	10	40	74	**21 863**	**21 863**	**21 863**	**21 863**	**21 863**
Sum	60		605	**174 167**	174 188	174 309	**174 167**	174 190
#Best				**60**	59	55	**60**	59

Table 3: Results for the VRPRDL benchmark of Ozbaygin *et al.* (2017)

			BPC		ΣCost of MathHeu configurations			
Instances	#	$\|N\|$	Σveh	ΣCost	MH	MH+BS	MH+SPP	MH+SPP+BS
1–5	5	15	19	**5 450**	**5 450**	**5 450**	**5 450**	**5 450**
6–10	5	20	20	**5 604**	**5 604**	**5 604**	**5 604**	**5 604**
11–20	10	30	52	**15 128**	**15 128**	**15 128**	**15 128**	**15 128**
21–30	10	60	83	**26 800**	**26 800**	**26 800**	**26 800**	**26 800**
31–40	10	120	128	38 107‡	37 263	37 349	37 234	**37 232**
41–50_v1	10	40	87	**27 996**	**27 996**	**27 996**	**27 996**	**27 996**
41–50_v2	10	40	67	**20 958**	**20 958**	20 962	20 962	21 029
Sum	60		456	140.043	139.199	139.289	**139.174**	139.239
#Best				53	57	54	**58**	56

Table 4: Results for the VRPHRDL benchmark of Ozbaygin *et al.* (2017)

algorithm solves all 60 instances for the VRPRDL and 53 of 60 instances for the VRPHRDL benchmark set to proven optimality. The entry marked with ‡ indicates that best-known solution values have been used here, because optimality could not be proven by the BPC algorithm for seven instances in this group. Again, the best solution values are marked in bold.

Comparing the four matheuristic configurations, `MH+SPP` performs best on both problem variants. For the VRPRDL, all optimal solutions could be found, while for the three other configurations the results are between 0.01 % and 0.04 % worse than the optimal solutions. Regarding the VRPHRDL, the four configurations improve the cost over the solutions provided in Tilk *et al.* (2020). `MH+SPP` contributes with the largest improvement (0.6 %). Moreover, seven new best-known solutions have been computed. Detailed results can be found in Appendix B.

4.4 Results for the VRPMTW

We compare the `MathHeu` configurations with the `ALNS` of Larsen and Pacino (2019) on the adapted Solomon benchmark instances for the VRPMTW provided by Belhaiza *et al.* (2014). The algorithm of Belhaiza *et al.* minimizes the sum of the total routing cost and cost of the used vehicles. Since vehicle costs are fairly large, we run Algorithm 1 twice, first to minimize the number of vehicles (which is then bounded), and second to minimize the routing cost, similar to Ropke and Pisinger (2006a). The first phase tries to decrease the number of vehicles by removing the smallest route from the solution each time a feasible solution is found.

We report the best solution found out of 10 runs with an overall time budget of 600 seconds for both phases per instance. Larsen and Pacino (2019) used the same time limit and number of runs. Moreover, they performed their computational experiments on an Intel Core i7-4790K @ 4.00GHz, which is 2.02 times faster than our processor (see PassMark-Software, 2020).

				`MathHeu` configurations							
		`ALNS`		`MH`		`MH+BS`		`MH+SPP`		`MH+SPP+BS`	
Group	#	Σveh	ΣCost	Σveh	ΣCost	Σveh	ΣCost	Σveh	ΣCost	Σveh	ΣCost
rm 1	8	73	21 831.5	73	21 849.0	73	21 830.8	**73**	**21 817.6**	73	21 825.3
rm 2	8	**16**	**21 477.3**	16	21 489.3	16	21 486.8	16	21 500.4	16	21 499.3
cm 1	8	86	25 821.1	84	25 754.3	83	25 750.5	83	25 609.3	**83**	**25 550.6**
cm 2	8	37	33 249.9	37	33 265.4	37	33 319.6	37	33 279.0	**37**	**33 245.0**
rcm 1	8	**82**	**25 747.4**	82	25 805.2	82	25 815.7	82	25 801.7	82	25 806.3
rcm 2	8	**16**	**21 854.0**	16	21 865.0	16	21 881.1	16	21 881.1	16	21 878.4
Sum	48	310	149 981.2	308	150 028.1	307	150 084.6	307	149 889.1	307	**149 804.9**
#Best			18		17		16		20		**24**

Table 5: Results for the VRPMTW benchmark of Belhaiza *et al.* (2014).

Table 5 shows aggregated results for the six groups of instances (R=random, C=clustered, or RC=partly random, partly clustered; series 1 and 2 with tight and wide time windows, respectively). Detailed results per instance are presented in Appendix C.

Comparing all five algorithms, the `ALNS` produces solutions with more vehicles than our `MathHeu` configurations, which is caused by the different performance in the group CM 1. Regarding the overall-cost objective, results over all five algorithms are rather similar (total differences are below 0.2 %). `MH+SPP+BS` performs best, it improves the results of the `ALNS` by 0.11 % and provides the most best-known solutions. Over the 48 instances, 23 new best solutions are found by (at least) one of the four `MathHeu` configurations.

4.5 Results for the VRPDO

For the VRPDO, we test the `MathHeu` configurations on the instances proposed by Dumez *et al.* (2020). The required service levels are given by $\beta_1 = 80\,\%$ and $\beta_2 = 90\,\%$. The time budget is set to 300 seconds for 100-customer instances and 2 000 seconds for 200-customer instances.

We rerun the `LNS` of Dumez *et al.* (2020) on all instances with the new time limit. The tests were performed on the same computer.

Recall that the VRPDO has a hierarchical objective. Thus, similar to Ropke and Pisinger (2006a), Algorithm 1 is run twice, first to minimize the number of vehicles (which is then bounded), and second to minimize the routing cost. The first phase systematically removes the smallest route from the current solution each time a feasible solution is found.

Table 6 presents the aggregated results. Each line refers to a group of instances with identical number of customers and options. Detailed results can be found in Appendix D.

				`MathHeu` configurations							
		`LNS`		`MH`		`MH+BS`		`MH+SPP`		`MH+SPP+BS`	
Group	#	Σveh	ΣCost	Σveh	ΣCost	Σveh	ΣCost	Σveh	ΣCost	Σveh	ΣCost
U100	10	**105**	**6 489.95**	105	6 558.55	105	6 612.34	105	6 528.21	105	6 548.26
U200	10	205	13 421.93	205	11 959.32	205	11 983.34	**205**	**11 768.85**	205	11 786.57
V100	10	104	7 031.37	104	7 078.08	104	7 070.62	**104**	**7 026.62**	104	7 052.34
V200	10	204	15 089.31	204	13 716.05	204	13 725.32	**203**	**13 741.99**	204	13 540.19
UBC100	10	40	3 624.20	**40**	**3 593.88**	40	3 657.44	40	3 599.04	40	3 612.59
UBC200	10	80	6 338.74	**80**	**6 062.81**	80	6 227.54	80	6 072.03	80	6 232.32
Sum	60	738	51 995.49	738	48 968.69	738	49 276.59	**737**	**48 736.75**	738	48 772.27
#Best			10		12		6		22		**23**

Table 6: Results for the VRPDO benchmark of Dumez *et al.* (2020)

Overall, the four `MathHeu` configurations clearly outperform the `LNS` that is also equipped with a SPP component which is only controlled with a much simpler manner compared to the new adaptive layer, see Section 4.1. Among the `MathHeu` configurations, `MH+SPP` produces the best results with 737 routes and the smallest routing costs. Indeed, `MH+SPP` is the only configuration that is able to find a solution with 20 vehicles on instance V_200_3. For all other instances the minimum number of vehicles is identical over all configurations. `MH+SPP` improves the results compared to `LNS` by 6.3 % on average. It finds the best-known solution on 22 of the 60 instances.

Configuration `MH+SPP+BS` performs very similar to `MH+SPP`. It finds one more best-known solution and the solution quality is less than 0.1 % worse. Moreover, the detailed results of Appendix D show that configuration `MH+SPP+BS` exclusively finds new best solutions for 14 instances. Overall, the four `MathHeu` configurations together provide 41 new best-known solutions.

4.6 Summary

To summarize the results for the four GVRPTW variants, we group all instances according to different criteria to further assess the performance of the matheuristic configurations. To this end, we report the gap to the best-known solution and the number of best-known solutions found. Formally, the gap (in percent) is computed as $100 \cdot (UB - BKS)/BKS$, where BKS is the best-known solution value from the literature and UB is the best solution value found by the respective algorithm.

For the VRPDO, we consider only instances for which all four configurations achieve a solution with the same number of vehicles. As a consequence, the instance `V_200_3` is disregarded.

We group the instances according to the VRP variant, the number of customers per route in the BKS, and the number of options per customer.

Table 7 shows aggregated results regarding the performance per group: The first column indicates the category used for grouping the instances, the second column gives the value defining the group, and the third column shows the number of instances in that group. The next four columns report the average gap and the last four columns report the number of best-known solutions found. In each line, the smallest average gap and the largest number of BKSs found are highlighted in bold. Regarding the different problem variants, all configurations except

Results grouped by		#	% Gap				# BKS			
			MH	MH+BS	MH+SPP	MH+SPP+BS	MH	MH+BS	MH+SPP	MH+SPP+BS
Variant	GVRPTW	20	0.27	0.47	**0.25**	0.27	**11**	8	10	**11**
	VRPRDL	60	0.01	0.04	**0.00**	0.01	59	55	**60**	59
	VRPHRDL	60	0.04	0.08	**0.03**	0.08	57	54	**58**	56
	VRPMTW	48	0.32	0.35	0.23	**0.18**	17	16	20	**24**
	VRPDO	59	1.28	2.24	**0.73**	1.31	12	6	22	**23**
Customers per Route	[0.6[	77	**0.00**	0.03	**0.00**	**0.00**	**77**	75	**77**	**77**
	[6,10)	83	0.37	0.45	0.21	**0.20**	50	43	56	**59**
	[10,20)	44	1.06	1.31	**0.41**	0.55	11	11	**24**	23
	[20,∞)	43	**0.50**	1.60	0.61	1.30	**18**	10	12	14
Options per Customer	[1,2)	32	0.95	0.95	0.46	**0.44**	4	9	11	**14**
	[2,3)	52	0.96	2.08	**0.65**	1.30	19	9	21	**22**
	[3,4)	97	0.04	0.09	0.03	**0.02**	90	83	**91**	**91**
	[4,∞)	66	0.22	0.28	**0.14**	0.19	43	38	**46**	**46**
Total		247	0.40	0.67	**0.25**	0.39	156	139	169	**173**

Table 7: Performance of the four matheuristic configurations

`MH+BS` perform rather similarly on the instances for the GVRPTW, VRPRDL, and VRPHRDL. For VRPMTW instances, `MH+SPP+BS` is clearly the best configuration, and for the VRPDO instances, `MH+SPP` is the best. In particular, regarding '# BKS', the two configurations with the SPP component outperform their counterparts without SPP component on the VRPMTW and VRPDO.

The rather bad performance of the BS component on VRPDO instances can be attributed to the additional synchronizations constraints only present in this variant. These constraints imply that a larger set of resources has to be taken into account in the label-setting algorithm. As a result, the practical difficulty of the SPPRC increases substantially, making the BS component too time-consuming relative to the improvements obtained with BS.

When instances are grouped according to route length, we can clearly see that the two exact components (i.e., `MH+SPP+BS`) are beneficial for the route lengths between 6 and 20 customers. There is nearly no difference between configurations for routes with 6 or less customers, while the exact components worsen the results for more than 20 customers.

When grouped according to the number of options per request, it seems that gaps decrease when the number of options per customer rises. However, this trend is not clear cut. Other instance characteristics seem to be more important.

Summarizing, configuration `MH+BS` performs worst, while both configurations `MH+SPP` and `MH+SPP+BS` are very competitive. The former produces the best gaps, while the latter provides the most BKS. Both configurations are also complementing each other well, because the overlap between the 169 and 173 BKS (see last two columns of Table 7) is only 145 instances. We can also conclude that the BS component is only beneficial in combination with the SPP component.

5 Conclusions

The GVRPTW is the archetypal problem in vehicle routing that combines time window constraints and delivery options. It exists in several variants under different names, sometimes focussing on particular aspects (VRPRDL, VRPMTW). We consider additional inter-route resource constraints giving rise to the so-called vehicle routing problem with delivery options (VRPDO), which captures two very important practical side constraints that can be found in last-mile delivery, e.g., in postal and package delivery applications. On the one hand, customers value the delivery options (home delivery, delivery to a locker or shop etc.) differently so that service-level constraints become relevant. On the other hand, delivery options of different customers may share limited capacities, e.g., restricted space for storage in lockers and shops from where customers retrieve their packages. The VRPDO adds these constraints to the basic GVRPTW.

In this paper, we presented a new LNS-based matheuristic that can cope with GVRPTW variants, in particular also with the most general and more involved VRPDO. The new matheuristic has an adaptive layer that controls the size of the destroy operation and the use of two (optional) exact components. The latter exact components allow an improvement of solutions when the standard LNS process stalls. The first component utilizes a MIP solver and SPP formulation that selects routes from a larger pool of potential routes. The second component uses BS neighborhood that we adapt for the use in a multiple vehicle context with capacity, time window, and inter-route constraints. Both components are, compared to a single LNS iteration, very time-consuming. Hence, the adaptive layer very carefully controls how often and when the exact components are invoked.

The primary focus of our research is the evaluation of the adaptive layer as well as the two exact components. We compare four different configurations of the new matheuristic (`MH`, `MH+BS`, `MH+SPP`, and `MH+SPP+BS`, i.e., with and without SPP and BS component) among each other and against state-of-the-art algorithms for the GVRPTW, VRPRDL, VRPHRDL, VRPMTW, and VRPDO. The experiments are conducted on standard benchmarks for these problems and have led to the following insights:

- A key success factor of the underlying LNS is the tremendous number of iterations that can be performed thanks to the small destroy moves, while the search is diversified by larger destroy moves if necessary. The way the adaptive layer selects small and fast or large destroy moves already makes the new matheuristic superior to an older LNS (see Section 4.5).
- The two configurations with the SPP component outperform their counterparts without SPP component regarding average gaps to best-known solutions (BKS) from the literature and the number of BKS computed (Table 7).
- The configuration with the BS component alone (`MH+BS`) is inferior. In particular, for the VRPDO, the presence of many inter-route constraints that must be handled within the BS neighborhood exploration makes the BS component too time-consuming (see Section 4.5). However, when combined with SPP the resulting configuration `MH+SPP+BS` is competitive.
- The practical difficulty of the VRPDO is well reflected in the results delivered by all matheuristic configurations. Average gaps of 0.73 % for the VRPDO are much bigger than gaps for the other GVRPTW variants which fall below 0.25 % (Table 7).
- Compared to state-of-the-art metaheuristics from the literature, the new matheuristic is much faster (factor 7 for the GVRPTW; factor 2 for the VRPMTW, see Sections 4.2 and 4.4).
- For the VRPRDL (VRPHRDL), the comparison against a recent exact branch-price-and-cut algorithm shows that configuration `MH+SPP` finds all (all except one) known optimal solutions in a fraction of the computation time available for the exact algorithm. In addition, all configurations together provide new best-known solutions for all instances that were not

solved to proven optimality by the branch-price-and-cut algorithm (Section 4.3).
- Overall, without manual problem-specific parameter tuning of the matheuristic, we provide 81 new best-known solutions for the GVRPTW, VRPHRDL, VRPMTW, and VRPDO.

Acknowledgement

This research was supported by the Agence Nationale de la Recherche (ANR) under grant ANR-17-CE22-0015 and Deutsche Forschungsgemeinschaft (DFG) under grant IR 122/8-1. This support is gratefully acknowledged.

References

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.

Belhaiza, S., Hansen, P., and Laporte, G. (2014). A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. *Computers & Operations Research*, **52**, 269–281.

Belhaiza, S., M'Hallah, R., and Brahim, G. B. (2017). A new hybrid genetic variable neighborhood search heuristic for the vehicle routing problem with multiple time windows. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1319–1326. IEEE.

Cattaruzza, D., Absi, N., Feillet, D., and González-Feliu, J. (2017). Vehicle routing problems for city logistics. *EURO Journal on Transportation and Logistics*, **6**(1), 51–79.

Christiaens, J. and Vanden Berghe, G. (2016). A fresh ruin and recreate implementation for the capacitated vehicle routing problem. Technical report, KU Leuven, Ghent, Belgium.

Christiaens, J. and Vanden Berghe, G. (2019). Slack induction by string removals for vehicle routing problems. *Transportation Science*, **54**, 417–433.

Demir, E., Bektaş, T., and Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, **223**(2), 346–359.

Desaulniers, G., Madsen, O. B. G., and Ropke, S. (2014). The vehicle routing problem with time windows. In P. Toth and D. Vigo, editors, *Vehicle Routing*, chapter 5, pages 119–159. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Drexl, M. (2012). Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, **46**(3), 297–316.

Dumez, D., Lehuédé, F., and Péton, O. (2020). A large neighborhood search approach to the vehicle routing problem with delivery options. HAL archives ouvertes hal-02452252. `https://hal.archives-ouvertes.fr/hal-02452252`.

Favaretto, D., Moretti, E., and Pellegrini, P. (2007). Ant colony system for a VRP with multiple time windows and multiple visits. *Journal of Interdisciplinary Mathematics*, **10**(2), 263–284.

Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR*, **45**(4), 239–256.

Foster, B. A. and Ryan, D. M. (1976). An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, **27**(2), 367–384.

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

Hempsch, C. and Irnich, S. (2008). Vehicle routing problems with inter-tour resource constraints. In B. L. Golden, R. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 421–444. Springer Nature.

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

Hoogeboom, M., Dullaert, W., Lai, D., and Vigo, D. (2020). Efficient neighborhood evaluations for the vehicle routing problem with multiple time windows. *Transportation Science*, **54**(2), 400–416.

IBM (2018). *CPLEX*. `https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer`.

Irnich, S. (2008a). Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

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

Irnich, S. (2008c). A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. *INFORMS Journal on Computing*, **20**(2), 270–287.

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

Larsen, R. and Pacino, D. (2019). Fast delta evaluation for the vehicle routing problem with multiple time windows. arXiv preprint arXiv:1905.04114.

Lombard, A., Tamayo-Giraldo, S., and Fontane, F. (2018). Vehicle routing problem with roaming delivery locations and stochastic travel times (VRPRDL-S). *Transportation Research Procedia*, **30**, 167–177.

Moccia, L., Cordeau, J.-F., and Laporte, G. (2012). An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. *Journal of the Operational Research Society*, **63**(2), 232–244.

Nagata, Y. and Bräysy, O. (2009). A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, **37**(5), 333–338.

Ozbaygin, G., Karasan, O. E., Savelsbergh, M., and Yaman, H. (2017). A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. *Transportation Research Part B: Methodological*, **100**, 115–137.

PassMark-Software (2020). CPU benchmarks. `https://www.cpubenchmark.net`.

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. (2019). Large neighborhood search. In *Handbook of Metaheuristics*, pages 99–127. Springer.

Reyes, D., Savelsbergh, M., and Toriello, A. (2017). Vehicle routing with roaming delivery locations. *Transportation Research Part C: Emerging Technologies*, **80**, 71–91.

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.

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.

Savelsbergh, M. W. P. (1985). Local search in routing problems with time windows. *Annals of Operations Research*, **4**(1), 285–305.

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

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.

Tellez, O., Vercraene, S., Lehuédé, F., Péton, O., and Monteiro, T. (2018). The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity. *Transportation Research Part C: Emerging Technologies*, **91**, 99–123.

Tilk, C. and Irnich, S. (2016). Dynamic programming for the minimum tour duration problem. *Transportation Science*, **51**(2), 549–565.

Tilk, C., Olkis, K., and Irnich, S. (2020). The last-mile vehicle routing problem with delivery options. Technical Report LM-2020-06, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

Toffolo, T. A. M., Vidal, T., and Wauters, T. (2019). Heuristics for vehicle routing problems: Sequence or set optimization? *Computers & Operations Research*, **105**, 118–131.

Toth, P. and Vigo, D. (2014). *Vehicle routing: problems, methods, and applications*. SIAM.

Turkeš, R., Sörensen, K., Hvattum, L. M., Barrena, E., Chentli, H., Coelho, L., Dayarian, I., Grimault, A., Gullhav, A., Iris, Ç., Keskin, M., Kiefer, A., Lusby, R., Mauri, G., Monroy-Licht, M., Parragh, S., Riquelme-Rodríguez, J.-P., Santini, A., Santos Vinicius, G. M., and Thomas, C. (2019). Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search. Working paper d/2019/1169/002, University of Antwerp.

Yıldırım, U. M. and Çatay, B. (2015). An ant colony-based matheuristic approach for solving a class of vehicle routing problems. In *International Conference on Computational Logistics (ILCL)*, pages 105–119. Springer.

A Detailed results for the GVRPTW instances

Instance	MH				MH+BS				MH+SPP				MH+SPP+BS				Time [s]
	Average		Best of 5		Average		Best of 5		Average		Best of 5		Average		Best of 5		
	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	
i-030-04-08	4	3 497.00	4	3 497	4	3 497.00	4	3 497	4	3 497.00	4	3 497	4	3 497.00	4	3 497	5
i-030-08-12	4.6	3 152.29	4	2 796	4.4	3 066.71	4	2 796	4.4	2 771.14	4	2 796	4.2	2 802.00	4	2 796	5
i-040-04-08	6	3 841.67	6	3 811	6	3 837.00	6	3 811	6	3 818.00	6	3 811	6	3 823.67	6	3 811	8
i-040-08-12	6.7	4 094.83	6	3 768	6.7	4 100.00	6	3 768	6.7	3 820.00	6	3 768	6.5	3 908.67	6	3 768	8
i-050-04-08	8	5 485.17	8	5 447	8	5 458.83	8	5 447	8	5 451.33	8	5 439	8	5 479.80	8	5 447	11
i-050-08-12	8	4 070.33	8	4 034	8	4 055.00	8	4 034	8	4 043.67	8	4 034	8	4 034.00	8	4 034	11
i-060-04-08	8	5 936.40	8	5 908	8	5 972.40	8	5 926	8	5 936.80	8	5 919	8	5 941.40	8	5 926	14
i-060-08-12	8	4 327.40	8	4 303	8	4 396.40	8	4 303	8	4 413.20	8	4 303	8	4 395.60	8	4 332	14
i-070-04-08	10	6 266.80	10	6 246	10	6 273.80	10	6 246	10	6 249.60	10	6 224	10	6 254.00	10	6 246	22
i-070-08-12	10	4 734.20	10	4 644	10	4 803.20	10	4 644	10	4 728.40	10	4 644	10	4 780.20	10	4 644	22
i-080-04-08	12	7 428.20	12	7 390	12	7 472.00	12	7 394	12	7 427.00	12	7 394	12	7 463.40	12	7 394	27
i-080-08-12	12	5 718.40	12	5 686	12	5 802.80	12	5 661	12	5 752.20	12	5 692	12	5 772.00	12	5 661	27
i-090-04-08	11.4	7 514.80	11	7 187	11.4	7 418.20	11	7 182	11.4	7 431.40	11	7 187	11.6	7 508.80	11	7 215	32
i-090-08-12	11	6 113.00	11	5 903	11	5 937.20	11	5 869	11	5 938.40	11	5 830	11	5 956.40	11	5 808	32
i-100-04-08	14	7 394.80	14	7 308	14	7 407.60	14	7 349	14	7 377.60	14	7 295	14	7 346.20	14	7 295	38
i-100-08-12	14	6 643.20	14	6 546	14	6 648.40	14	6 546	14	6 677.60	14	6 585	14	6 677.40	14	6 606	38
i-110-04-08	16	8 768.60	16	8 696	16	8 766.60	16	8 720	16	8 726.00	16	8 711	16	8 709.60	16	8 687	45
i-110-08-12	16	6 379.80	16	6 249	16	6 611.80	16	6 487	16	6 406.20	16	6 338	16	6 394.60	16	6 310	45
i-120-04-08	15	8 356.60	15	8 344	15	8 402.60	15	8 357	15	8 355.20	15	8 344	15	8 356.80	15	8 344	60
i-120-08-12	15	7 018.80	15	6 829	15	7 005.80	15	6 829	15	6 859.80	15	6 774	15	6 877.40	15	6 774	60
Total	209.6	116 742.29	208	114 592	209.5	116 933.35	208	114 866	209.5	115 680.54	208	114 585	209.3	115 978.93	208	114 595	524

Table 8: Detailed results for the GVRPTW instances

B Detailed results for the VRPRDL and VRPHRDL instances

Instance	MH				MH+BS				MH+SPP				MH+SPP+BS			
	Average		Best of 5		Average		Best of 5		Average		Best of 5		Average		Best of 5	
	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost
instance_0	4.0	901.0	4	901	4.0	901.0	4	901	4.0	901.0	4	901	4.0	901.0	4	901
instance_1	5.0	1 286.0	5	1 286	5.0	1 286.0	5	1 286	5.0	1 286.0	5	1 286	5.0	1 286.0	5	1 286
instance_2	4.0	991.0	4	991	4.0	991.0	4	991	4.0	991.0	4	991	4.0	991.0	4	991
instance_3	5.0	1 062.0	5	1 062	5.0	1 062.0	5	1 062	5.0	1 062.0	5	1 062	5.0	1 062.0	5	1 062
instance_4	6.0	1 832.0	6	1 832	6.0	1 832.0	6	1 832	6.0	1 832.0	6	1 832	6.0	1 832.0	6	1 832
instance_5	5.0	1 294.0	5	1 294	5.0	1 294.0	5	1 294	5.0	1 294.0	5	1 294	5.0	1 294.0	5	1 294
instance_6	4.0	1 155.0	4	1 155	4.0	1 155.0	4	1 155	4.0	1 155.0	4	1 155	4.0	1 155.0	4	1 155
instance_7	6.0	1 455.0	6	1 455	6.0	1 455.0	6	1 455	6.0	1 455.0	6	1 455	6.0	1 455.0	6	1 455
instance_8	5.0	1 260.0	5	1 260	5.0	1 260.0	5	1 260	5.0	1 260.0	5	1 260	5.0	1 260.0	5	1 260
instance_9	7.0	1 684.0	7	1 684	7.0	1 684.0	7	1 684	7.0	1 684.0	7	1 684	7.0	1 684.0	7	1 684
instance_10	7.0	1 922.0	7	1 922	7.0	1 922.0	7	1 922	7.0	1 922.0	7	1 922	7.0	1 922.0	7	1 922
instance_11	8.0	2 324.0	8	2 324	8.0	2 324.0	8	2 324	8.0	2 324.0	8	2 324	8.0	2 324.0	8	2 324
instance_12	6.0	1 747.0	6	1 747	6.0	1 747.0	6	1 747	6.0	1 747.0	6	1 747	6.0	1 747.0	6	1 747
instance_13	6.0	1 273.0	6	1 273	6.0	1 273.0	6	1 273	6.0	1 273.0	6	1 273	6.0	1 273.0	6	1 273
instance_14	6.0	1 694.0	6	1 694	6.0	1 694.0	6	1 694	6.0	1 694.0	6	1 694	6.0	1 694.0	6	1 694
instance_15	7.0	1 938.0	7	1 938	7.0	1 938.0	7	1 938	7.0	1 938.0	7	1 938	7.0	1 938.0	7	1 938
instance_16	8.0	1 965.0	8	1 965	8.0	1 965.0	8	1 965	8.0	1 965.0	8	1 965	8.0	1 965.0	8	1 965
instance_17	7.0	1 827.0	7	1 827	7.0	1 827.0	7	1 827	7.0	1 827.0	7	1 827	7.0	1 827.0	7	1 827
instance_18	7.0	2 083.0	7	2 083	7.0	2 083.0	7	2 083	7.0	2 083.0	7	2 083	7.0	2 083.0	7	2 083
instance_19	6.0	1 822.0	6	1 822	6.0	1 822.0	6	1 822	6.0	1 822.0	6	1 822	6.0	1 822.0	6	1 822
instance_20	13.0	3 761.0	13	3 761	13.0	3 761.0	13	3 761	13.0	3 761.0	13	3 761	13.0	3 761.0	13	3 761
instance_21	10.0	2 828.0	10	2 828	10.0	2 828.0	10	2 828	10.0	2 828.0	10	2 828	10.0	2 828.0	10	2 828
instance_22	16.0	4 440.0	16	4 440	16.0	4 440.0	16	4 440	16.0	4 440.0	16	4 440	16.0	4 440.0	16	4 440
instance_23	11.0	3 378.0	11	3 378	11.0	3 378.0	11	3 378	11.0	3 378.0	11	3 378	11.0	3 378.0	11	3 378
instance_24	11.0	3 161.0	11	3 161	11.0	3 162.5	11	3 161	11.0	3 161.4	11	3 161	11.0	3 161.0	11	3 161
instance_25	16.0	4 536.0	16	4 536	16.0	4 536.0	16	4 536	16.0	4 536.0	16	4 536	16.0	4 536.0	16	4 536
instance_26	10.0	2 865.0	10	2 865	10.0	2 865.0	10	2 865	10.0	2 865.0	10	2 865	10.0	2 865.0	10	2 865
instance_27	14.0	4 173.0	14	4 173	14.0	4 173.0	14	4 173	14.0	4 173.0	14	4 173	14.0	4 173.0	14	4 173
instance_28	14.0	3 964.0	14	3 964	14.0	3 964.0	14	3 964	14.0	3 964.0	14	3 964	14.0	3 964.0	14	3 964
instance_29	14.0	4 107.0	14	4 107	14.0	4 107.0	14	4 107	14.0	4 107.0	14	4 107	14.0	4 107.0	14	4 107
instance_30	17.0	4 935.0	17	4 935	17.0	4 935.0	17	4 935	17.0	4 935.0	17	4 935	17.0	4 935.0	17	4 935
instance_31	18.0	5 269.3	18	5 258	18.0	5 311.0	18	5 267	18.0	5 258.0	18	5 258	18.0	5 261.6	18	5 258
instance_32	18.0	5 061.0	18	5 061	18.0	5 061.6	18	5 061	18.0	5 061.0	18	5 061	18.0	5 061.0	18	5 061

instance_33	17.0	5 248.2	17	5 218	17.0	5 259.8	17	5 220	17.0	5 218.0	17	5 218	17.0	5 218.0	17	5 218
instance_34	20.0	5 521.0	20	5 498	20.0	5 525.2	20	5 498	20.0	5 511.8	20	5 498	20.0	5 529.5	20	5 521
instance_35	22.0	6 498.0	22	6 498	22.0	6 621.0	22	6 621	22.0	6 498.0	22	6 498	22.0	6 503.3	22	6 498
instance_36	17.0	4 850.5	17	4 830	17.0	4 856.3	17	4 830	17.0	4 833.3	17	4 830	17.0	4 832.5	17	4 830
instance_37	22.0	5 618.4	22	5 604	22.0	5 631.3	22	5 604	22.0	5 604.0	22	5 604	22.0	5 604.0	22	5 604
instance_38	21.0	5 842.0	21	5 841	21.0	5 863.8	21	5 841	21.0	5 841.0	21	5 841	21.0	5 845.8	21	5 841
instance_39	17.8	5 194.3	17	5 016	17.3	5 076.0	17	4 999	17.0	5 004.5	17	4 995	17.0	5 002.2	17	4 995
41_v1	10.0	3 203.0	10	3 203	10.0	3 211.2	10	3 203	10.0	3 203.0	10	3 203	10.0	3 211.2	10	3 203
42_v1	9.0	2 799.0	9	2 799	9.0	2 799.0	9	2 799	9.0	2 799.0	9	2 799	9.0	2 799.0	9	2 799
43_v1	8.0	2 604.6	8	2 603	8.0	2 607.0	8	2 607	8.0	2 606.0	8	2 603	8.0	2 605.4	8	2 603
44_v1	7.0	2 261.0	7	2 261	7.0	2 261.0	7	2 261	7.0	2 261.0	7	2 261	7.0	2 261.0	7	2 261
45_v1	10.0	3 217.0	10	3 217	10.0	3 217.0	10	3 217	10.0	3 217.0	10	3 217	10.0	3 217.0	10	3 217
46_v1	9.0	2 805.0	9	2 805	9.0	2 805.0	9	2 805	9.0	2 805.0	9	2 805	9.0	2 805.0	9	2 805
47_v1	10.0	3 339.0	10	3 339	10.0	3 339.0	10	3 339	10.0	3 339.0	10	3 339	10.0	3 339.0	10	3 339
48_v1	10.0	3 325.0	10	3 325	10.0	3 325.0	10	3 325	10.0	3 325.0	10	3 325	10.0	3 325.0	10	3 325
49_v1	11.0	3 534.0	11	3 534	11.0	3 534.0	11	3 534	11.0	3 534.0	11	3 534	11.0	3 534.0	11	3 534
50_v1	10.0	2 752.0	10	2 752	10.0	2 752.0	10	2 752	10.0	2 752.0	10	2 752	10.0	2 752.0	10	2 752
41_v2	7.0	2 133.0	7	2 133	7.0	2 133.4	7	2 133	7.0	2 133.0	7	2 133	7.0	2 133.0	7	2 133
42_v2	6.0	1 946.0	6	1 946	6.0	1 946.0	6	1 946	6.0	1 946.0	6	1 946	6.0	1 946.0	6	1 946
43_v2	8.0	1 966.0	8	1 966	8.0	1 966.0	8	1 966	8.0	1 966.0	8	1 966	8.0	1 966.0	8	1 966
44_v2	6.0	1 610.0	6	1 610	6.0	1 610.0	6	1 610	6.0	1 610.0	6	1 610	6.0	1 610.0	6	1 610
45_v2	8.0	2 478.0	8	2 478	8.0	2 478.0	8	2 478	8.0	2 478.0	8	2 478	8.0	2 478.0	8	2 478
46_v2	8.0	2 469.0	8	2 469	8.0	2 469.0	8	2 469	8.0	2 469.0	8	2 469	8.0	2 469.0	8	2 469
47_v2	7.0	1 946.0	7	1 946	7.0	1 946.0	7	1 946	7.0	1 946.0	7	1 946	7.0	1 946.0	7	1 946
48_v2	8.0	2 380.0	8	2 380	8.0	2 380.0	8	2 380	8.0	2 380.0	8	2 380	8.0	2 380.0	8	2 380
49_v2	8.0	2 492.0	8	2 492	8.0	2 492.5	8	2 492	8.0	2 492.0	8	2 492	8.0	2 492.0	8	2 492
50_v2	8.0	2 443.0	8	2 443	8.0	2 444.8	8	2 443	8.0	2 443.0	8	2 443	8.0	2 443.0	8	2 443
total	605.8	174 468.2	605	174 188	605.3	174 586.2	605	174 309	605.0	174 197.0	605	174 167	605.0	174 232.4	605	174 190

Table 9: Detailed results for the VRPRDL instances

Instance	MH				MH+BS				MH+SPP				MH+SPP+BS			
	Average		Best of 5		Average		Best of 5		Average		Best of 5		Average		Best of 5	
	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost
instance_0	3.0	773.0	3	773	3.0	773.0	3	773	3.0	773.0	3	773	3.0	773.0	3	773
instance_1	4.0	1 065.0	4	1 065	4.0	1 065.0	4	1 065	4.0	1 065.0	4	1 065	4.0	1 065.0	4	1 065
instance_2	3.0	988.0	3	988	3.0	988.0	3	988	3.0	988.0	3	988	3.0	988.0	3	988
instance_3	3.0	914.0	3	914	3.0	914.0	3	914	3.0	914.0	3	914	3.0	914.0	3	914
instance_4	6.0	1 710.0	6	1 710	6.0	1 710.0	6	1 710	6.0	1 710.0	6	1 710	6.0	1 710.0	6	1 710
instance_5	4.0	1 099.0	4	1 099	4.0	1 099.0	4	1 099	4.0	1 099.0	4	1 099	4.0	1 099.0	4	1 099
instance_6	3.0	996.0	3	996	3.0	996.0	3	996	3.0	996.0	3	996	3.0	996.0	3	996
instance_7	5.0	1 346.0	5	1 346	5.0	1 346.0	5	1 346	5.0	1 346.0	5	1 346	5.0	1 346.0	5	1 346
instance_8	4.0	997.0	4	997	4.0	997.0	4	997	4.0	997.0	4	997	4.0	997.0	4	997
instance_9	4.0	1 166.0	4	1 166	4.0	1 166.0	4	1 166	4.0	1 166.0	4	1 166	4.0	1 166.0	4	1 166
instance_10	5.0	1 595.8	5	1 587	5.0	1 595.8	5	1 587	5.0	1 595.8	5	1 587	5.0	1 595.6	5	1 587
instance_11	6.0	1 808.0	6	1 808	6.0	1 808.0	6	1 808	6.0	1 808.0	6	1 808	6.0	1 808.0	6	1 808
instance_12	6.0	1 563.0	6	1 563	6.0	1 563.0	6	1 563	6.0	1 563.0	6	1 563	6.0	1 563.0	6	1 563
instance_13	4.0	1 058.0	4	1 058	4.0	1 058.0	4	1 058	4.0	1 058.0	4	1 058	4.0	1 058.0	4	1 058
instance_14	5.0	1 347.0	5	1 347	5.0	1 347.0	5	1 347	5.0	1 347.0	5	1 347	5.0	1 347.0	5	1 347
instance_15	5.0	1 517.0	5	1 517	5.0	1 517.0	5	1 517	5.0	1 517.0	5	1 517	5.0	1 517.0	5	1 517
instance_16	5.0	1 445.0	5	1 445	5.0	1 445.0	5	1 445	5.0	1 445.0	5	1 445	5.0	1 445.0	5	1 445
instance_17	5.0	1 627.0	5	1 627	5.0	1 627.0	5	1 627	5.0	1 627.0	5	1 627	5.0	1 627.0	5	1 627
instance_18	5.0	1 461.0	5	1 461	5.0	1 461.0	5	1 461	5.0	1 461.0	5	1 461	5.0	1 461.0	5	1 461
instance_19	6.0	1 715.0	6	1 715	6.0	1 715.0	6	1 715	6.0	1 715.0	6	1 715	6.0	1 715.0	6	1 715
instance_20	8.0	2 597.6	8	2 580	8.0	2 585.6	8	2 580	8.0	2 580.0	8	2 580	8.0	2 580.0	8	2 580
instance_21	7.0	2 206.0	7	2 206	7.0	2 206.0	7	2 206	7.0	2 206.0	7	2 206	7.0	2 206.0	7	2 206
instance_22	10.0	3 363.0	10	3 363	10.0	3 363.0	10	3 363	10.0	3 363.0	10	3 363	10.0	3 363.0	10	3 363
instance_23	8.0	2 569.0	8	2 569	8.0	2 569.0	8	2 569	8.0	2 569.0	8	2 569	8.0	2 569.0	8	2 569
instance_24	8.0	2 383.6	8	2 378	8.0	2 383.0	8	2 378	8.0	2 380.8	8	2 378	8.0	2 389.2	8	2 378
instance_25	9.0	2 845.0	9	2 845	9.0	2 845.3	9	2 845	9.0	2 845.0	9	2 845	9.0	2 845.0	9	2 845
instance_26	8.0	2 518.0	8	2 518	8.0	2 518.3	8	2 518	8.0	2 518.0	8	2 518	8.0	2 518.0	8	2 518
instance_27	8.0	2 758.0	8	2 758	8.0	2 758.0	8	2 758	8.0	2 758.0	8	2 758	8.0	2 758.0	8	2 758
instance_28	9.0	2 892.0	9	2 892	9.0	2 892.0	9	2 892	9.0	2 892.0	9	2 892	9.0	2 892.0	9	2 892
instance_29	8.0	2 691.0	8	2 691	8.0	2 691.0	8	2 691	8.0	2 691.0	8	2 691	8.0	2 691.0	8	2 691
instance_30	12.0	3 666.0	12	3 666	12.0	3 765.2	12	3 666	12.0	3 666.0	12	3 666	12.0	3 726.0	12	3 666
instance_31	13.0	3 885.2	13	3 885	13.0	3 885.4	13	3 885	13.0	3 885.0	13	3 885	13.0	3 885.0	13	3 885
instance_32	13.0	3 554.2	13	3 543	13.0	3 588.2	13	3 543	13.0	3 543.2	13	3 543	13.0	3 557.0	13	3 543
instance_33	12.0	3 767.4	12	3 705	12.0	3 783.4	12	3 783	12.2	3 783.6	12	3 694	12.0	3 751.3	12	3 715
instance_34	11.0	3 184.0	11	3 184	11.0	3 184.8	11	3 184	11.0	3 184.0	11	3 184	11.0	3 174.8	11	3 138

instance_35	15.0	4 286.6	15	4 273	15.0	4 378.5	15	4 279	15.0	4 273.0	15	4 273	15.0	4 302.8	15	4 278
instance_36	10.0	3 225.0	10	3 217	10.5	3 369.8	10	3 217	10.0	3 217.3	10	3 217	10.0	3 217.3	10	3 217
instance_37	14.0	3 935.0	14	3 935	14.0	3 939.3	14	3 937	14.0	3 935.8	14	3 935	14.0	3 936.5	14	3 935
instance_38	15.0	4 300.0	15	4 300	15.0	4 335.3	15	4 300	15.0	4 300.0	15	4 300	15.0	4 300.0	15	4 300
instance_39	13.0	3 556.2	13	3 555	13.0	3 557.3	13	3 555	13.0	3 549.5	13	3 537	13.0	3 556.0	13	3 555
41_v1	8.0	2 662.0	8	2 662	8.0	2 662.0	8	2 662	8.0	2 662.0	8	2 662	8.0	2 662.0	8	2 662
42_v1	8.0	2 610.0	8	2 610	8.0	2 610.0	8	2 610	8.0	2 610.0	8	2 610	8.0	2 610.0	8	2 610
43_v1	7.0	2 260.0	7	2 260	7.0	2 260.0	7	2 260	7.0	2 260.0	7	2 260	7.0	2 260.0	7	2 260
44_v1	7.0	2 147.0	7	2 147	7.0	2 147.0	7	2 147	7.0	2 147.0	7	2 147	7.0	2 147.0	7	2 147
45_v1	10.0	3 172.0	10	3 172	10.0	3 172.0	10	3 172	10.0	3 172.0	10	3 172	10.0	3 172.0	10	3 172
46_v1	8.0	2 616.0	8	2 616	8.0	2 616.0	8	2 616	8.0	2 616.0	8	2 616	8.0	2 616.0	8	2 616
47_v1	9.0	3 010.0	9	3 010	9.0	3 010.0	9	3 010	9.0	3 010.0	9	3 010	9.0	3 010.0	9	3 010
48_v1	10.0	3 278.0	10	3 278	10.0	3 278.0	10	3 278	10.0	3 278.0	10	3 278	10.0	3 278.0	10	3 278
49_v1	11.0	3 514.0	11	3 514	11.0	3 514.0	11	3 514	11.0	3 514.0	11	3 514	11.0	3 514.0	11	3 514
50_v1	9.0	2 727.0	9	2 727	9.0	2 727.0	9	2 727	9.0	2 727.0	9	2 727	9.0	2 727.0	9	2 727
41_v2	6.6	2 070.0	6	1 998	6.6	2 109.2	6	1 998	6.2	2 019.0	6	1 998	6.8	2 096.2	6	2 069
42_v2	6.0	1 930.5	6	1 927	6.0	1 931.0	6	1 931	6.0	1 931.0	6	1 931	6.0	1 930.2	6	1 927
43_v2	6.0	1 830.0	6	1 830	6.0	1 830.0	6	1 830	6.0	1 830.0	6	1 830	6.0	1 830.0	6	1 830
44_v2	5.0	1 478.0	5	1 478	5.0	1 478.0	5	1 478	5.0	1 478.0	5	1 478	5.0	1 478.0	5	1 478
45_v2	8.0	2 466.0	8	2 466	8.0	2 466.0	8	2 466	8.0	2 466.0	8	2 466	8.0	2 466.0	8	2 466
46_v2	8.0	2 388.0	8	2 388	8.0	2 388.0	8	2 388	8.0	2 388.0	8	2 388	8.0	2 388.0	8	2 388
47_v2	6.0	1 848.0	6	1 848	6.0	1 854.4	6	1 848	6.0	1 854.4	6	1 848	6.0	1 860.8	6	1 848
48_v2	7.0	2 264.0	7	2 264	7.0	2 266.4	7	2 264	7.0	2 265.0	7	2 264	7.0	2 265.5	7	2 264
49_v2	8.0	2 457.0	8	2 457	8.0	2 457.0	8	2 457	8.0	2 457.0	8	2 457	8.0	2 457.0	8	2 457
50_v2	7.5	2 419.3	7	2 302	7.5	2 363.5	7	2 302	7.5	2 329.8	7	2 302	7.0	2 302.0	7	2 302
total	457.1	139 520.4	456	139 199	457.6	139 928.4	456	139 289	456.9	139 345.1	456	139 174	456.8	139 478.1	456	139 239

Table 10: Detailed results for the VRPHRDL instances

C Detailed results for the VRPMTW instances

Instance	MH				MH+BS				MH+SPP				MH+SPP+BS			
	Average		Best of 5		Average		Best of 5		Average		Best of 5		Average		Best of 5	
	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost
rm101	10.0	2 978.6	10	2 972.3	10.0	2 983.0	10	2 969.7	10.0	2 973.2	10	2 967.8	10.0	2 976.6	10	2 967.8
rm102	9.6	2 844.5	9	2 730.8	9.4	2 813.4	9	2 715.3	9.2	2 759.0	9	2 703.9	9.3	2 791.8	9	2 711.6
rm103	9.0	2 689.1	9	2 682.5	9.0	2 690.9	9	2 682.5	9.0	2 688.8	9	2 682.5	9.0	2 694.2	9	2 682.5
rm104	9.0	2 692.4	9	2 688.3	9.0	2 693.1	9	2 688.3	9.0	2 691.5	9	2 688.3	9.0	2 692.8	9	2 688.3
rm105	9.0	2 684.9	9	2 682.0	9.0	2 687.2	9	2 682.0	9.0	2 684.8	9	2 682.0	9.0	2 684.2	9	2 682.0
rm106	9.0	2 701.9	9	2 700.0	9.0	2 702.2	9	2 700.0	9.0	2 701.3	9	2 700.0	9.0	2 702.0	9	2 700.0
rm107	9.0	2 679.4	9	2 679.4	9.0	2 682.4	9	2 679.4	9.0	2 681.8	9	2 679.4	9.0	2 679.4	9	2 679.4
rm108	9.0	2 716.7	9	2 713.7	9.0	2 716.8	9	2 713.7	9.0	2 717.2	9	2 713.7	9.0	2 718.2	9	2 713.7
rm201	2.0	2 776.4	2	2 756.2	2.0	2 787.9	2	2 752.9	2.0	2 779.2	2	2 761.7	2.0	2 782.0	2	2 759.0
rm202	2.0	2 691.4	2	2 682.6	2.0	2 691.7	2	2 683.5	2.0	2 698.5	2	2 685.6	2.0	2 694.0	2	2 687.1
rm203	2.0	2 689.4	2	2 679.2	2.0	2 685.3	2	2 679.2	2.0	2 690.1	2	2 679.2	2.0	2 689.6	2	2 679.7
rm204	2.0	2 677.1	2	2 672.8	2.0	2 681.5	2	2 672.8	2.0	2 678.2	2	2 672.8	2.0	2 678.4	2	2 672.8
rm205	2.0	2 673.0	2	2 671.3	2.0	2 676.3	2	2 671.0	2.0	2 675.8	2	2 671.0	2.0	2 675.2	2	2 671.3
rm206	2.0	2 684.9	2	2 679.0	2.0	2 688.8	2	2 679.0	2.0	2 686.9	2	2 679.0	2.0	2 686.1	2	2 679.0
rm207	2.0	2 683.7	2	2 674.4	2.0	2 685.6	2	2 674.4	2.0	2 681.9	2	2 674.4	2.0	2 686.6	2	2 674.7
rm208	2.0	2 679.1	2	2 673.9	2.0	2 677.3	2	2 673.9	2.0	2 680.1	2	2 676.7	2.0	2 679.3	2	2 675.7
cm101	10.0	3 181.9	10	3 112.2	10.0	3 161.1	10	3 120.8	10.0	3 175.0	10	3 091.1	10.0	3 115.4	10	3 060.2
cm102	12.0	3 451.2	12	3 445.3	11.9	3 451.2	11	3 370.0	11.6	3 406.4	11	3 326.3	11.8	3 442.1	11	3 305.4
cm103	11.0	3 472.1	11	3 416.3	11.0	3 483.4	11	3 420.7	11.0	3 453.4	11	3 415.3	11.0	3 463.8	11	3 429.3
cm104	13.0	3 883.0	13	3 859.6	13.3	3 936.2	13	3 873.4	13.0	3 866.7	13	3 859.6	13.2	3 903.5	13	3 859.6
cm105	10.0	3 017.8	10	2 999.3	10.0	3 020.5	10	2 999.3	10.0	3 014.1	10	2 999.3	10.0	3 013.6	10	2 999.3
cm106	9.2	3 012.5	9	2 975.6	9.3	3 029.9	9	3 008.6	9.3	3 005.4	9	2 976.1	9.3	2 998.6	9	2 953.1
cm107	10.0	3 077.1	10	3 077.1	10.0	3 077.1	10	3 077.1	10.0	3 077.1	10	3 077.1	10.0	3 077.5	10	3 077.1
cm108	9.0	2 896.9	9	2 868.8	9.0	2 910.0	9	2 880.6	9.0	2 885.5	9	2 864.5	9.0	2 885.8	9	2 866.7
cm201	5.0	4 416.8	5	4 365.6	5.0	4 441.6	5	4 395.9	5.0	4 414.8	5	4 376.1	5.0	4 433.9	5	4 371.1
cm202	6.0	4 996.8	6	4 982.4	6.0	5 007.7	6	4 991.4	6.0	5 001.6	6	4 982.4	6.0	4 999.7	6	4 982.4
cm203	5.0	4 468.2	5	4 438.6	5.0	4 474.1	5	4 452.9	5.0	4 466.4	5	4 446.8	5.0	4 475.4	5	4 432.8
cm204	5.0	4 337.4	5	4 317.5	5.0	4 344.0	5	4 317.5	5.0	4 340.3	5	4 317.5	5.0	4 344.3	5	4 317.5
cm205	4.0	3 859.9	4	3 832.1	4.0	3 842.4	4	3 822.4	4.0	3 843.4	4	3 820.6	4.0	3 863.4	4	3 800.7
cm206	4.0	3 723.2	4	3 696.0	4.0	3 712.4	4	3 699.9	4.0	3 724.6	4	3 696.0	4.0	3 707.6	4	3 696.0
cm207	4.0	3 954.2	4	3 919.4	4.0	3 944.2	4	3 925.9	4.0	3 954.9	4	3 925.9	4.0	3 949.6	4	3 930.7
cm208	4.0	3 725.8	4	3 713.9	4.0	3 736.2	4	3 713.9	4.0	3 735.2	4	3 713.9	4.0	3 736.8	4	3 713.9
rcm101	10.0	3 076.8	10	3 074.0	10.0	3 080.8	10	3 074.0	10.0	3 079.4	10	3 074.0	10.0	3 077.8	10	3 074.0

rcm102	10.0	3 122.9	10	3 121.6	10.2	3 171.2	10	3 121.6	10.1	3 146.0	10	3 121.6	10.1	3 159.9	10	3 121.6
rcm103	10.0	3 129.0	10	3 120.9	10.0	3 125.4	10	3 120.9	10.0	3 130.5	10	3 120.9	10.0	3 129.5	10	3 120.9
rcm104	10.0	3 127.3	10	3 124.3	10.0	3 131.0	10	3 124.3	10.0	3 125.5	10	3 124.3	10.0	3 126.3	10	3 124.3
rcm105	10.0	3 167.8	10	3 165.4	10.0	3 169.5	10	3 165.4	10.0	3 166.8	10	3 165.4	10.0	3 169.9	10	3 165.4
rcm106	10.0	3 179.1	10	3 165.7	10.1	3 194.6	10	3 167.7	10.0	3 174.7	10	3 165.7	10.0	3 173.4	10	3 165.7
rcm107	11.0	3 495.3	11	3 490.8	11.0	3 495.9	11	3 496.4	11.0	3 493.8	11	3 490.8	11.0	3 496.1	11	3 495.4
rcm108	11.0	3 553.8	11	3 542.5	11.0	3 548.0	11	3 545.4	11.0	3 544.3	11	3 539.0	11.0	3 545.8	11	3 539.0
rcm201	2.0	2 757.1	2	2 716.5	2.0	2 749.5	2	2 735.8	2.0	2 754.9	2	2 698.3	2.0	2 750.6	2	2 727.4
rcm202	2.0	2 745.0	2	2 719.5	2.0	2 745.5	2	2 727.2	2.0	2 748.6	2	2 732.5	2.0	2 748.7	2	2 722.9
rcm203	2.0	2 729.9	2	2 704.8	2.0	2 725.1	2	2 704.8	2.0	2 744.8	2	2 710.7	2.0	2 733.9	2	2 704.8
rcm204	2.0	2 702.5	2	2 692.6	2.0	2 699.4	2	2 692.2	2.0	2 696.5	2	2 692.6	2.0	2 704.1	2	2 692.2
rcm205	2.0	2 721.6	2	2 711.6	2.0	2 726.3	2	2 718.9	2.0	2 737.1	2	2 723.6	2.0	2 729.3	2	2 718.8
rcm206	2.0	2 744.7	2	2 732.4	2.0	2 755.7	2	2 721.9	2.0	2 759.2	2	2 736.9	2.0	2 743.8	2	2 721.9
rcm207	2.9	3 659.3	2	2 865.0	2.8	3 555.1	2	2 857.6	2.9	3 659.4	2	2 863.7	2.8	3 559.4	2	2 867.9
rcm208	2.0	2 733.1	2	2 722.7	2.0	2 735.1	2	2 722.7	2.0	2 734.7	2	2 722.7	2.0	2 732.1	2	2 722.7
total	309.7	151 662.3	308	150 028.1	310.1	151 723.5	307	150 084.6	309.1	151 529.2	307	149 889.1	309.5	151 501.9	307	149 804.9

Table 11: Detailed results for the VRPMTW instances

D Detailed results for the VRPDO instances

Instance	MH				MH+BS				MH+SPP				MH+SPP+BS			
	Average		Best of 5		Average		Best of 5		Average		Best of 5		Average		Best of 5	
	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost
U_100_1	11	483.314	11	481.815	11	486.620	11	482.775	11	482.422	11	481.815	11	481.475	11	479.972
U_100_2	10	700.868	10	690.492	10	706.882	10	703.562	10	691.408	10	672.935	10	704.306	10	691.132
U_100_3	11	638.635	11	638.344	11	637.781	11	637.302	11	632.362	11	631.513	11	634.724	11	630.020
U_100_4	10	615.068	10	612.776	10	615.839	10	614.972	10	610.459	10	607.131	10	614.789	10	609.721
U_100_5	10	684.479	10	680.541	10	708.342	10	701.200	10	677.526	10	673.312	10	679.947	10	673.312
U_100_6	10	599.256	10	592.423	10	611.721	10	601.699	10	604.854	10	598.608	10	602.237	10	598.608
U_100_7	11	756.634	11	747.047	11	755.025	11	749.348	11	764.625	11	763.735	11	766.904	11	763.644
U_100_8	11	860.448	11	856.522	11	856.120	11	852.734	11	850.435	11	843.291	11	850.559	11	845.885
U_100_9	10	689.242	10	683.954	10	695.653	10	693.102	10	683.792	10	680.248	10	684.387	10	682.765
U_100_10	11	575.522	11	574.633	11	579.580	11	575.644	11	576.283	11	575.623	11	574.621	11	573.197
V_100_1	11	655.611	11	654.423	11	648.805	11	647.566	11	647.696	11	647.566	11	647.696	11	647.566
V_100_2	10	865.081	10	857.681	10	865.582	10	855.525	10	858.881	10	851.125	10	865.677	10	848.567
V_100_3	10	711.580	10	707.802	10	715.417	10	700.693	10	694.383	10	686.649	10	708.965	10	706.331
V_100_4	10	601.748	10	597.021	10	605.573	10	596.840	10	599.580	10	594.605	10	598.477	10	594.605
V_100_5	10	779.856	10	778.419	10	795.838	10	790.631	10	783.311	10	771.775	10	809.732	10	802.718
V_100_6	11	600.044	11	599.459	11	599.385	11	599.275	11	597.288	11	597.288	11	598.156	11	597.288
V_100_7	11	705.915	11	705.063	11	706.822	11	703.775	11	708.458	11	707.928	11	707.009	11	703.341
V_100_8	11	749.876	11	749.419	11	753.808	11	747.435	11	751.669	11	750.096	11	748.446	11	745.349
V_100_9	10	815.874	10	811.910	10	808.097	10	805.638	10	809.362	10	804.279	10	808.749	10	800.959
V_100_10	10	617.940	10	616.879	10	624.708	10	623.237	10	617.417	10	615.309	10	608.978	10	605.617
UBC_100_1	4	368.636	4	365.802	4	370.097	4	368.237	4	370.097	4	368.237	4	370.097	4	368.237
UBC_100_2	4	344.624	4	344.624	4	349.995	4	346.316	4	353.569	4	353.098	4	351.769	4	350.970
UBC_100_3	4	323.464	4	323.464	4	323.464	4	323.464	4	323.464	4	323.464	4	323.464	4	323.464
UBC_100_4	4	337.950	4	334.615	4	334.924	4	334.615	4	337.950	4	334.615	4	336.029	4	334.615
UBC_100_5	4	371.537	4	371.037	4	392.966	4	391.884	4	380.563	4	371.549	4	381.405	4	371.549
UBC_100_6	4	355.332	4	349.580	4	381.508	4	379.375	4	352.329	4	346.906	4	371.882	4	363.770
UBC_100_7	4	337.902	4	337.902	4	337.902	4	337.902	4	337.902	4	337.902	4	337.902	4	337.902
UBC_100_8	4	422.822	4	422.483	4	427.742	4	423.280	4	424.281	4	422.483	4	418.019	4	415.075
UBC_100_9	4	391.634	4	388.042	4	399.755	4	396.029	4	387.614	4	384.456	4	401.030	4	384.456
UBC_100_10	4	356.334	4	356.334	4	358.823	4	356.334	4	358.823	4	356.334	4	364.269	4	362.556
Total	249	17 317.224	249	17 230.506	249	17 454.774	249	17 340.389	249	17 268.802	249	17 153.875	249	17 351.697	249	17 213.191

Table 12: Detailed results for the VRPDO instances with 100 customers

Instance	MH				MH+BS				MH+SPP				MH+SPP+BS			
	Average		Best of 5		Average		Best of 5		Average		Best of 5		Average		Best of 5	
	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost	#veh	Cost
U_200_1	21	1 548.342	21	1 542.480	21	1 539.278	21	1 536.110	21	1 513.882	21	1 508.560	21	1 502.832	21	1 491.370
U_200_2	21	1 216.688	21	1 208.020	21	1 217.644	21	1 214.840	21	1 197.432	21	1 195.000	21	1 202.872	21	1 189.360
U_200_3	20	1 322.138	20	1 310.720	20	1 332.844	20	1 324.300	20	1 297.886	20	1 289.180	20	1 296.944	20	1 289.470
U_200_4	21	940.705	21	931.995	21	942.423	21	939.911	21	928.217	21	924.914	21	924.582	21	920.695
U_200_5	21	1 078.336	21	1 072.840	21	1 071.700	21	1 056.720	21	1 056.238	21	1 052.940	21	1 052.728	21	1 050.600
U_200_6	20	1 109.626	20	1 106.350	20	1 119.048	20	1 106.790	20	1 089.182	20	1 081.080	20	1 092.318	20	1 087.790
U_200_7	21	1 023.036	21	1 007.640	21	1 039.078	21	1 035.930	21	1 017.442	21	1 001.940	21	1 012.292	21	1 007.830
U_200_8	20	1 116.908	20	1 095.710	20	1 122.330	20	1 105.120	20	1 098.650	20	1 093.050	20	1 090.720	20	1 079.740
U_200_9	20	1 203.062	20	1 197.170	20	1 202.216	20	1 183.060	20	1 186.462	20	1 177.170	20	1 199.858	20	1 191.320
U_200_10	20	1 490.590	20	1 486.390	20	1 485.150	20	1 480.560	20	1 468.650	20	1 445.020	20	1 485.614	20	1 478.390
V_200_1	21	1 301.110	21	1 293.370	21	1 308.064	21	1 307.260	21	1 301.780	21	1 292.040	21	1 300.722	21	1 293.090
V_200_2	20	1 247.108	20	1 241.740	20	1 265.440	20	1 246.820	20	1 222.038	20	1 212.850	20	1 214.104	20	1 203.660
V_200_3	21	1 262.810	21	1 255.860	21	1 260.366	21	1 250.380	20.6	1 308.970	20	1 397.310	21	1 250.494	21	1 243.620
V_200_4	21	1 379.662	21	1 368.870	21	1 384.654	21	1 378.370	21	1 359.564	21	1 349.480	21	1 342.916	21	1 329.730
V_200_5	20	1 416.110	20	1 409.910	20	1 404.168	20	1 398.440	20.4	1 375.512	20	1 364.600	20	1 390.838	20	1 369.240
V_200_6	21	1 453.872	21	1 446.090	21	1 436.040	21	1 431.780	21	1 464.726	21	1 453.410	21	1 453.182	21	1 445.760
V_200_7	20	1 172.212	20	1 166.020	20	1 166.680	20	1 158.900	20	1 169.826	20	1 166.340	20	1 166.274	20	1 163.270
V_200_8	20	1 515.420	20	1 499.910	20	1 496.576	20	1 489.010	20	1 502.948	20	1 491.970	20	1 493.640	20	1 486.380
V_200_9	20	1 640.758	20	1 636.770	20	1 658.764	20	1 652.390	20	1 638.112	20	1 623.120	20	1 630.906	20	1 623.930
V_200_10	20	1 407.380	20	1 397.510	20	1 422.134	20	1 411.970	20	1 399.784	20	1 390.870	20	1 391.236	20	1 381.510
UBC_200_1	8	584.186	8	570.262	8	595.963	8	573.345	8	587.010	8	572.463	8	595.690	8	590.562
UBC_200_2	8	492.103	8	491.967	8	508.878	8	508.495	8	492.755	8	491.523	8	495.136	8	493.489
UBC_200_3	8	790.370	8	781.196	8	813.213	8	801.096	8	787.989	8	783.611	8	819.289	8	802.842
UBC_200_4	8	654.003	8	647.939	8	674.314	8	669.280	8	643.298	8	639.340	8	650.038	8	647.167
UBC_200_5	8	619.365	8	616.843	8	629.376	8	619.177	8	622.453	8	620.446	8	639.134	8	633.661
UBC_200_6	8	620.324	8	620.040	8	649.853	8	644.267	8	626.438	8	620.751	8	638.475	8	627.195
UBC_200_7	8	591.024	8	589.950	8	616.753	8	592.136	8	608.325	8	590.601	8	616.665	8	607.539
UBC_200_8	8	616.004	8	613.413	8	626.750	8	620.689	8	622.741	8	608.578	8	649.081	8	642.490
UBC_200_9	8	532.771	8	525.495	8	560.208	8	559.022	8	548.009	8	537.647	8	545.697	8	525.656
UBC_200_10	8	611.626	8	605.709	8	640.371	8	640.036	8	620.200	8	607.067	8	667.705	8	661.724
Total	489	31 957.649	489	31 738.179	489	32 190.275	489	31 936.204	489	31 756.519	488	31 582.871	489	31 811.982	489	31 559.080

Table 13: Detailed results for the VRPDO instances with 200 customers