Supplementary Material

for

Robust Optimization of a Broad Class of Heterogeneous Vehicle Routing Problems under Demand Uncertainty

Anirudh Subramanyam[1], Panagiotis P. Repoussis[2,3], and Chrysanthos E. Gounaris[1]

[1]Carnegie Mellon University, Pittsburgh PA, United States

[2]Stevens Institute of Technology, Hoboken NJ, United States

[3]Athens University of Economics and Business, Athens, Greece

This document contains material that has been omitted from the main paper for the sake of readability and/or brevity. The document is structured as follows. Section 1 contains proofs of all propositions from the main text. In Section 2, we provide details on our implementation of a hybrid Iterated Local Search metaheuristic, while Section 3 provides details on our implementation of a hybrid Adaptive Memory Programming metaheuristic. Finally, in Section 4, we present summary tables of best found solutions from all our computational runs. We remark that a comprehensive spreadsheet with detailed results from all our runs has also been made available online.

Note: All equation labels refer to those in the main paper.

1 Propositions with Proofs

Proposition 1. *Suppose R is a given route and $S \subseteq V_C$ is the set of customers visited on R. Then, the worst-case load of route R over the uncertainty sets $\mathcal{Q}_B$, $\mathcal{Q}_F$, $\mathcal{Q}_E$, $\mathcal{Q}_G$ and $\mathcal{Q}_D$ is as follows.*

$\mathcal{Q}$	$\max_{q \in \mathcal{Q}} \sum_{i \in R} q_i$	
$\mathcal{Q}_B$	$\sum_{i \in S} \bar{q}_i - \sum_{l=1}^{L} \max \left\{ 0, \sum_{i \in S \cap B_l} (\bar{q}_i - \underline{q}_i) - \left(b_l - \sum_{i \in B_l} \underline{q}_i \right) \right\}$	(7)
$\mathcal{Q}_F$	$\sum_{i \in S} q_i^0 - \min \left\{ \sum_{f=1}^{F} \left\lvert \sum_{i \in S} \Psi_{if} - \lambda \right\rvert + \beta F \lvert \lambda \rvert : \lambda \in \left\{ 0, \sum_{i \in S} \Psi_{if_{\ell^+}}, \sum_{i \in S} \Psi_{if_{\ell^-}} \right\} \right\}$,	(8)

where $f_1, \ldots f_F$ *represents an ordering of the factors such that* $\sum_{i \in S} \Psi_{if_1} \geq \ldots \geq \sum_{i \in S} \Psi_{if_F}$, *and* $\ell^+ = \lceil (1+\beta)F/2 \rceil$, $\ell^- = \max\{1, \lceil (1-\beta)F/2 \rceil\}$.

$$\mathcal{Q}_E \qquad \sum_{i \in S} q_i^0 + \left\| \sum_{i \in S} \Sigma_i^{1/2} \right\|_2, \tag{9}$$

where $\Sigma_i^{1/2}$ *denotes the* i^{th} *column of* $\Sigma^{1/2}$ *and* $\|\cdot\|_2$ *denotes the* ℓ_2*-norm of a vector.*

$$\mathcal{Q}_G \qquad \sum_{i \in S} q_i^0 + \sum_{\ell=1}^{\min\{|S|, \lfloor \Gamma \rfloor\}} \hat{q}_{g_\ell} + \lambda, \tag{10}$$

where $g_1, \ldots g_{|S|}$ *represents an ordering of the customers in* S *such that* $\hat{q}_{g_1} \geq \ldots \geq \hat{q}_{g_{|S|}}$, *and* $\lambda = (\Gamma - \lfloor \Gamma \rfloor)\, \hat{q}_{g_{\lfloor \Gamma \rfloor + 1}}$, *if* $|S| \geq \lfloor \Gamma \rfloor + 1$, *and* 0 *otherwise.*

$$\mathcal{Q}_D \qquad \max \left\{ \sum_{i \in S} q_i^{(d)} : d \in \{1, \ldots, D\} \right\} \tag{11}$$

Proof. The validity of the expressions for $\mathcal{Q} = \mathcal{Q}_B$ and $\mathcal{Q} = \mathcal{Q}_F$ has been shown in [5].

Suppose that $\mathcal{Q} = \mathcal{Q}_E$. In this case, the worst-case problem (1) can be reformulated as follows:

$$\sum_{i \in S} q_i^0 + \max_{\xi \in \mathbb{R}^n} \left\{ \sum_{i \in S} \xi^\top \Sigma_i^{1/2} : \xi^\top \xi \leq 1 \right\}.$$

The above maximization problem is a convex optimization problem that satisfies Slater's constraint qualification (e.g., $0 \in \mathbb{R}^n$ is strictly feasible). Therefore, the Karush-Kuhn-Tucker conditions are both necessary and sufficient to characterize its optimal solution. This leads to expression (9).

Suppose that $\mathcal{Q} = \mathcal{Q}_G$. In this case, the worst-case problem (1) can be reformulated as follows:

$$\sum_{i \in S} q_i^0 + \max_{\xi \in \mathbb{R}^n_+} \left\{ \sum_{i \in S} \hat{q}_i \xi_i : \xi_i \leq 1 \; \forall i = 1, \ldots, n, \quad \sum_{i=1}^n \xi_i \leq \Gamma \right\}.$$

The above maximization problem is an instance of a *fractional knapsack* problem with n items, each with unit weight; the value of item i is $\hat{q}_i \mathbb{1}_{[i \in S]}$. It is well known [2] that this problem can be solved by a greedy algorithm that considers the items in non-increasing order of their values per unit weight, i.e., in non-increasing order of the $\hat{q}$ values of the items in S. A straightforward application of this greedy algorithm leads to expression (10).

Suppose now that $\mathcal{Q} = \mathcal{Q}_D$. Since $\mathcal{Q}_D$ is a bounded polytope and the objective function of the worst-case problem (1) is linear, the optimal solution is attained at a vertex of $\mathcal{Q}_D$. This leads to expression (11), since the set of vertices of $\mathcal{Q}_D$ is a subset of the D points that parametrize it. □

Proposition 2. *The time and storage complexities shown in Table 1 can be achieved for computing the worst-case load of a route that visits a set of customers S. Here, "incremental update" refers to the complexity of updating the worst-case load when a single customer is added to or removed from S.*

Table 1. Time and storage complexities for computing the worst-case load of a vehicle route.

$\mathcal{Q}$	From scratch	Incremental update	
	Time	Time	Storage
$\mathcal{Q}_B$	$\mathcal{O}(\lvert S\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(L)$
$\mathcal{Q}_F$	$\mathcal{O}(\lvert S\rvert\, F + F\log F)$	$\mathcal{O}(F\log F)$	$\mathcal{O}(F)$
$\mathcal{Q}_E$ (axis-parallel)	$\mathcal{O}(\lvert S\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\mathcal{Q}_E$ (general)	$\mathcal{O}(\lvert S\rvert\, n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
$\mathcal{Q}_G$	$\mathcal{O}(\lvert S\rvert)$	$\mathcal{O}(\log(\lvert S\rvert))^{\text{a}}$	$\mathcal{O}(n)^{\text{b}}$
$\mathcal{Q}_D$	$\mathcal{O}(\lvert S\rvert\, D)$	$\mathcal{O}(D)$	$\mathcal{O}(D)$

[a,b]These can be improved to $\mathcal{O}(\log(\Gamma))$ and $\mathcal{O}(\Gamma)$, respectively, if only additions to S are considered.

Proof. • Consider $\mathcal{Q} = \mathcal{Q}_B$. The time complexity for the "from scratch" computation follows directly from expression (7). To enable incremental updates, we use $\mathcal{O}(L)$ storage to keep track of the following quantities: $\pi = \sum_{i\in S} \bar{q}_i$, $\rho_l = \sum_{i\in S\cap B_l} \left(\bar{q}_i - \underline{q}_i\right) - \left(b_l - \sum_{i\in B_l} \underline{q}_i\right)$ for all $l = 1,\ldots,L$, and $z = \max_{q\in\mathcal{Q}} \sum_{i\in S} q_i$. Initially, when $S = \emptyset$, we have $(\pi, \rho_l, z) = (0, -(b_l - \sum_{i\in B_l} \underline{q}_i), 0)$. Now, suppose that we have updated this data structure to reflect the worst-case load of an "old" route visiting the customer set S, and that we would like to calculate the worst-case load of a "new" route visiting the customer set $S' = S\cup\{j\}$, where customer j participates in the budget l_j (i.e., $j \in B_{l_j}$). For this, we can perform the following update in $\mathcal{O}(1)$ time: $\pi^{\text{new}} \leftarrow \pi^{\text{old}} + \bar{q}_j$, $\rho_{l_j}^{\text{new}} \leftarrow \rho_{l_j}^{\text{old}} + (\bar{q}_j - \underline{q}_j)$, $z^{\text{new}} \leftarrow z^{\text{old}} + (\pi^{\text{new}} - \pi^{\text{old}}) - ([\rho_{l_j}^{\text{new}}]_+ - [\rho_{l_j}^{\text{old}}]_+)$, where $[\cdot]_+ = \max\{\cdot, 0\}$. If customer j does not participate in any budget, then we would repeat the same steps except ρ_l is not updated. A similar update applies when $S' = S \setminus \{j\}$ for some $j \in S$.

• Consider $\mathcal{Q} = \mathcal{Q}_F$. The time complexity for the "from scratch" computation follows from the $\mathcal{O}(|S|)$ time to calculate $\sum_{i\in S} \Psi_{if}$ for each of the F factors and the $\mathcal{O}(F\log F)$ time to compute the ordering $f_1,\ldots,f_F$. To enable incremental updates, we use $\mathcal{O}(F)$ storage to keep track of the

following quantities: $\pi = \sum_{i \in S} q_i^0$, $\rho_f = \sum_{i \in S} \Psi_{if}$ for all $f = 1, \ldots, F$, and $z = \max_{q \in \mathcal{Q}} \sum_{i \in S} q_i$. Initially, when $S = \emptyset$, we have $(\pi, \rho_f, z) = (0, 0, 0)$. To calculate the worst-case load of a route visiting the customer set $S' = S \cup \{j\}$, we can perform the following update: $\pi^{\text{new}} \leftarrow \pi^{\text{old}} + q_j^0$, $\rho_f^{\text{new}} \leftarrow \rho_f^{\text{old}} + \Psi_{jf}$ for each $f = 1, \ldots, F$, and $z^{\text{new}} \leftarrow \pi^{\text{new}} + \sum_{g=1}^{F} \xi_g^\star \rho_{f_g}^{\text{new}}$, where $f_1, \ldots, f_F$ is an updated ordering of the factors according to $\rho_{f_1}^{\text{new}} \geq \ldots \geq \rho_{f_F}^{\text{new}}$, and $\xi^\star \in \Xi_F$ is defined by first computing $P = \sum_{f=1}^{F} \mathbb{1}_{[\rho_f^{\text{new}} \geq 0]}$, $N = F - P$, $A = |P - N|$, $M = \min\{P, N\}$, $T = \lfloor (A - \lfloor \beta A \rfloor)/2 \rfloor$ and $Z = \lfloor \beta A \rfloor + T$, and then consulting the entries of the following table:

Case		$\xi^\star$
$P \geq N$	$T + Z = A$	$(e^M, e^Z, -e^T, -e^M)$
	$T + Z \neq A$	$(e^M, e^Z, +\beta F - \lfloor \beta F \rfloor, -e^T, -e^M)$
$P < N$	$T + Z = A$	$(e^M, e^T, -e^Z, -e^M)$
	$T + Z \neq A$	$(e^M, e^T, -\beta F + \lfloor \beta F \rfloor, -e^Z, -e^M)$

Here, $e^{\dim}$ denotes the vector of ones in $\mathbb{R}^{\dim}$. The overall time complexity of the update is $\mathcal{O}(F \log F)$ and is dictated by the sorting operation needed to compute the ordering $f_1, \ldots, f_F$.[1] A similar update applies when $S' = S \setminus \{j\}$ for some $j \in S$.

• Consider $\mathcal{Q} = \mathcal{Q}_E$, where $\mathcal{Q}_E$ is axis-parallel; that is, $\Sigma = \text{diag}(\sigma_1^2, \ldots, \sigma_n^2)$ is a diagonal matrix. In this case, the expression (9) simplifies to $\sum_{i \in S} q_i^0 + \sqrt{\sum_{i \in S} \sigma_i^2}$, which can be computed in $\mathcal{O}(|S|)$ time. To enable incremental updates, we use $\mathcal{O}(1)$ storage to keep track of the following quantities: $\pi = \sum_{i \in S} q_i^0$, $\rho = \sum_{i \in S} \sigma_i^2$ and $z = \max_{q \in \mathcal{Q}} \sum_{i \in S} q_i$. Initially, when $S = \emptyset$, we have $(\pi, \rho, z) = (0, 0, 0)$. To calculate the worst-case load of a route visiting the customer set $S' = S \cup \{j\}$, we can perform the following update in $\mathcal{O}(1)$ time: $\pi^{\text{new}} \leftarrow \pi^{\text{old}} + q_j^0$, $\rho^{\text{new}} \leftarrow \rho^{\text{old}} + \sigma_j^2$, $z^{\text{new}} \leftarrow \pi^{\text{new}} + \sqrt{\rho^{\text{new}}}$. A similar update applies when $S' = S \setminus \{j\}$ for some $j \in S$.

• Consider now $\mathcal{Q} = \mathcal{Q}_E$, where Σ is a general matrix. In this case, expression (9) can be written as $\sum_{i \in S} q_i^0 + \sqrt{\sum_{l=1}^{n} \left(\sum_{i \in S} \Sigma_{il}^{1/2} \right)^2}$, which can be computed in $\mathcal{O}(|S|\, n)$ time. To enable incremental updates, we use $\mathcal{O}(n)$ storage to keep track of the following quantities: $\pi = \sum_{i \in S} q_i^0$, $\rho_l = \sum_{i \in S} \Sigma_{il}^{1/2}$ for all $l = 1, \ldots, n$, and $z = \max_{q \in \mathcal{Q}} \sum_{i \in S} q_i$. When $S = \emptyset$, we have $(\pi, \rho_l, z) = (0, 0, 0)$. To calculate the worst-case load of a route visiting the customer set $S' = S \cup \{j\}$, we can perform

[1] In practice, the addition of a single customer j to a larger set S is unlikely to significantly change the ordering of the factors $f_1, \ldots, f_F$. We can take advantage of this by keeping track of the ordering and making use of some specialized sorting algorithm for "almost sorted" arrays, leading to even faster updates.

the following update in $\mathcal{O}(n)$ time: $\pi^{\text{new}} \leftarrow \pi^{\text{old}} + q_j^0$, $\rho_l^{\text{new}} \leftarrow \rho_l^{\text{old}} + \Sigma_{jl}^{1/2}$ for all $l = 1, \ldots, n$, $z^{\text{new}} \leftarrow \pi^{\text{new}} + \sqrt{\sum_{l=1}^{n} \left(\rho_l^{\text{new}}\right)^2}$. A similar update applies when $S' = S \setminus \{j\}$ for some $j \in S$.

• Consider $\mathcal{Q} = \mathcal{Q}_G$. An examination of expression (10) reveals that we do not need to sort the customers in S with respect to their $\hat{q}$ values. Instead, we only need to partition S around the customer with the $(\lfloor \Gamma \rfloor + 1)^{\text{th}}$ largest $\hat{q}$ value. This can be achieved in $\mathcal{O}(|S|)$ time with a *partition-based selection algorithm*, such as *quickselect* with an appropriate pivoting strategy [1]. To enable incremental updates, we use $\mathcal{O}(n)$ storage to keep track of the following quantities: $\pi = \sum_{i \in S} q_i^0$, $s = \min\{|S|, \lfloor \Gamma \rfloor\}$, array $h_+ = [\hat{q}_{g_1}, \ldots, \hat{q}_{g_s}]$ implemented as a binary min-heap, array $h_- = [\hat{q}_{g_{s+2}}, \ldots, \hat{q}_{g_{|S|}}]$ implemented as a binary max-heap, $\rho_+ =$ sum of entries of h_+, $\rho_0 = \hat{q}_{g_{s+1}}$, and $z = \max_{q \in \mathcal{Q}} \sum_{i \in S} q_i$, where we define $h_- = \emptyset$, if $|S| \leq \lfloor \Gamma \rfloor + 1$, and $\rho_0 = 0$, if $|S| \leq \lfloor \Gamma \rfloor$. Initially, when $S = \emptyset$, we have $(\pi, s, h_+, h_-, \rho_+, \rho_0, z) = (0, 0, \emptyset, \emptyset, 0, 0, 0)$. To calculate the worst-case load of a route visiting the customer set $S' = S \cup \{j\}$, we can perform the following update: if $s^{\text{old}} < \lfloor \Gamma \rfloor$, then $s^{\text{new}} \leftarrow s^{\text{old}} + 1$, $h_+^{\text{new}} \leftarrow h_+^{\text{old}}.\texttt{push}(\hat{q}_j)$, $\rho_+^{\text{new}} \leftarrow \rho_+^{\text{old}} + \hat{q}_j$; else if $\hat{q}_j \leq \rho_0$, then $h_-^{\text{new}} \leftarrow h_-^{\text{old}}.\texttt{push}(\hat{q}_j)$; else if $\hat{q}_j < h_+^{\text{old}}.\texttt{peek}()$, then $\rho_0^{\text{new}} \leftarrow \hat{q}_j$, $h_-^{\text{new}} \leftarrow h_-^{\text{old}}.\texttt{push}(\rho_0^{\text{old}})$; else, $h_+^{\text{new}} \leftarrow h_+^{\text{old}}.\texttt{delete\_min}().\texttt{push}(\hat{q}_j)$, $\rho_+^{\text{new}} \leftarrow \rho_+^{\text{old}} - h_+^{\text{old}}.\texttt{peek}() + \hat{q}_j$, $\rho_0^{\text{new}} \leftarrow h_+^{\text{old}}.\texttt{peek}()$ and $h_-^{\text{new}} \leftarrow h_-^{\text{old}}.\texttt{push}(\rho_0^{\text{old}})$. In all cases, we also update $\pi^{\text{new}} \leftarrow \pi^{\text{old}} + q_j^0$ and $z^{\text{new}} \leftarrow \pi^{\text{new}} + \rho_+^{\text{new}} + (\Gamma - \lfloor \Gamma \rfloor)\rho_0^{\text{new}}$. The overall time complexity of the update is $\mathcal{O}(\log(|S|))$ and is dictated by the insertion and deletion operations in the heaps h_+ and h_-. Note that h_- was not used in determining the value of z. In fact, it is used only when updating the worst-case load after a deletion has occurred (we omit the details for the sake of brevity). Consequently, if the quantities are maintained only by the addition of elements into initially empty structures, then h_+ is sufficient, and since its size is at most $\lfloor \Gamma \rfloor$, the overall time and storage complexities would be $\mathcal{O}(\log(\Gamma))$ and $\mathcal{O}(\Gamma)$, respectively. We also remark that merely querying the value of the worst-case load after a postulated addition or deletion of a customer can be achieved at $\mathcal{O}(1)$ time complexity, as one may forgo the heap updates in this case.

• Consider now $\mathcal{Q} = \mathcal{Q}_D$. The time complexity for the "from scratch" computation follows directly from expression (11). To enable incremental updates, we use $\mathcal{O}(D)$ storage to keep track of the following quantities: $\rho_d = \sum_{i \in S} q_i^{(d)}$ for all $d = 1, \ldots, D$, and $z = \max_{q \in \mathcal{Q}} \sum_{i \in S} q_i$. Initially, when $S = \emptyset$, we have $(\rho_d, z) = (0, 0)$. To calculate the worst-case load of a route visiting the customer set $S' = S \cup \{j\}$, we can perform the following update in $\mathcal{O}(D)$ time: $\rho_d^{\text{new}} \leftarrow \rho_d^{\text{old}} + q_j^{(d)}$ for all $d = 1, \ldots, D$, $z^{\text{new}} \leftarrow \max\{\rho_d^{\text{new}} : d = 1, \ldots, D\}$. A similar update applies when $S' = S \setminus \{j\}$

for some $j \in S$. $\square$

Proposition 3. *The feasible solutions of formulation (13) are in one-to-one correspondence with robust feasible solutions of the HVRP.*

Proof. Suppose $(\mathbf{R}, \boldsymbol{\kappa})$ is a robust feasible solution of the HVRP, i.e., it satisfies conditions **(C1)**, **(C2)** and **(C3r)**. Construct the solution (x, y) as follows: $y_{ik} = \sum_{h=1}^{H} \mathbb{1}_{[i \in R_h]} \mathbb{1}_{[k=\kappa_h]}$ and $x_{ijk} = \sum_{h=1}^{H} \sum_{l=0}^{|R_h|} \mathbb{1}_{[(i,j)=(r_{hl}, r_{hl+1})]} \mathbb{1}_{[k=\kappa_h]}$. We claim that (x, y) is a feasible solution of formulation (13). To see this, first observe that satisfaction of constraints (13b)–(13c) and (13f)–(13h) follows from the definition of a route (see Section 2 of the main paper) and from the fact that $(\mathbf{R}, \boldsymbol{\kappa})$ satisfies condition **(C1)**. Similarly, constraint (13d) is satisfied because $(\mathbf{R}, \boldsymbol{\kappa})$ satisfies condition **(C2)**. Constraints (13e) are satisfied because of the following reason. First, observe that we have:

$$\begin{aligned}
\sum_{(i,j) \in \delta_k(S)} x_{ijk} &\geq 2 \Big| \underbrace{h \in \{1, \ldots, H\} : \kappa_h = k \text{ and } S \cap R_h \neq \emptyset}_{:= H_k(S) = \text{index set of routes of type } k \text{ 'crossing' } S} \Big| \\
&= 2 \left\lceil \frac{1}{Q_k} \sum_{h \in H_k(S)} Q_k \right\rceil \\
&\geq 2 \left\lceil \frac{1}{Q_k} \max_{q \in \mathcal{Q}} \sum_{h \in H_k(S)} \sum_{i \in S \cap R_h} q_i \right\rceil = 2 \left\lceil \frac{1}{Q_k} \max_{q \in \mathcal{Q}} \sum_{i \in S \cap \left(\cup_{h \in H_k(S)} R_h \right)} q_i \right\rceil,
\end{aligned}$$

where the first inequality follows by construction of x while the second inequality follows because *(i)* $(\mathbf{R}, \boldsymbol{\kappa})$ satisfies condition (C3r) and *(ii)* the maximum operator is subadditive. Second, we have:

$$2 \sum_{i \in S} (1 - y_{ik}) = 2 \left| i \in S \setminus \left(\cup_{h \in H_k(S)} R_h \right) \right| \geq 2 \left\lceil \frac{1}{Q_k} \max_{q \in \mathcal{Q}} \sum_{i \in S \setminus \left(\cup_{h \in H_k(S)} R_h \right)} q_i \right\rceil,$$

where the equality follows by construction of y while the inequality follows because *(i)* each $i \in S \subseteq V_k$ satisfies $\max_{q \in \mathcal{Q}} q_i \leq Q_k$ and, *(ii)* the maximum and ceiling operators are subadditive. Finally, combining the above two expressions and using again the subadditivity of the maximum and ceiling operators shows that inequalities (13e) are satisfied.

Now, suppose that (x, y) is a feasible solution of formulation (13). Construct $(\mathbf{R}, \boldsymbol{\kappa})$ as follows: *(i)* $H \leftarrow 0$; *(ii)* for every $i \in V_C$, if $\sum_{k \in K} x_{0ik} = 1$ and $i \notin R_1, \ldots, R_H$, then set $H \leftarrow H + 1$ and define $\kappa_H = \sum_{k \in K} k \mathbb{1}_{[y_{ik}=1]}$ and R_H to be the cycle that passes through customer i in the subgraph of G induced by $\left\{ (i', j') \in E : x_{i'j'\kappa_H} = 1 \right\}$. We claim that $(\mathbf{R}, \boldsymbol{\kappa})$ is a robust feasible solution of the

HVRP. To see this, first observe that $(\mathbf{R}, \boldsymbol{\kappa})$ satisfies condition **(C1)** because of inequalities (13b)–(13c) and (13f)–(13h), and condition **(C2)** because of inequality (13d). To see that condition **(C3r)** is also satisfied for each route R_h, $h \in \{1, \ldots, H\}$: set $S = R_h$ and $k = \kappa_h$ in inequalities (13e). The left-hand side simplifies to 2 while the right-hand side simplifies to $2\lceil (1/Q_{\kappa_h}) \max_{q \in \mathcal{Q}} \sum_{i \in S} q_i \rceil$. Hence, we have $1 \geq \lceil (1/Q_{\kappa_h}) \max_{q \in \mathcal{Q}} \sum_{i \in S} q_i \rceil$, which implies that condition **(C3r)** is satisfied. □

2 Hybrid Iterated Local Search

Iterated Local Search (ILS), as the name implies, refers to the repeated application of local search to a current solution. The current solution may either be generated from scratch using a *construction heuristic* or by *perturbing* a locally optimal solution. We refer the reader to [6] for an introduction to this subject. Our specific hybrid ILS implementation for the robust HVRP is described in Algorithm 1.

Algorithm 1 consists of two phases: a *construction* phase (lines 3–5) and a *perturbation* phase (lines 6–17). The construction phase first constructs an initial solution (line 3) and then improves it using an efficient local search algorithm called *tabu search* (line 4). The perturbation phase iteratively perturbs this solution (line 8) and improves it using tabu search (line 9). The perturbation phase terminates if it fails to encounter a solution that is better than the best one found in the current iteration $(\mathbf{R}', \boldsymbol{\kappa}')$ for more than χ attempts (line 7). The ILS algorithm terminates after a pre-specified time limit t_{lim} is reached (line 2), at which point the best encountered solution $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right)$ is returned (line 19). The parameters η, ν, ζ and δ are used as inputs to the construction heuristic, tabu search and perturbation mechanisms, respectively, which we describe next.

Construction heuristic. The procedure CONSTRUCT SOLUTION(η) works by gradually inserting customers into an initially empty solution. At any given iteration, an empty route is first constructed for each vehicle type $k \in K$. To ensure that the fleet availability constraint **(C2)** is satisfied, this is done only if the number of routes of vehicle type k in the current partial solution is less than its available number m_k. Assuming this is the case, we keep adding unrouted customers to this route until it is no longer possible to do so (i.e., either because all customers have been routed or because the capacity condition **(C3r)** would be violated). Specifically, all customers that can be potentially added to the route are first inserted into a *restricted candidate list*; a random

Algorithm 1 Iterated local search.

Input: χ, η, ν, ζ, and δ (user-defined parameters)

Start timer t, $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right) \leftarrow (\emptyset, \emptyset)$
while $t < t_{\text{lim}}$ **do**
    $(\mathbf{R}, \boldsymbol{\kappa}) \leftarrow$ Construct Solution(η) ▷ Construction phase
    $(\mathbf{R}, \boldsymbol{\kappa}) \leftarrow$ Tabu Search$((\mathbf{R}, \boldsymbol{\kappa}), \nu, \zeta)$
    if $\bar{c}(\mathbf{R}, \boldsymbol{\kappa}) < \bar{c}\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right)$ **then** $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right) \leftarrow (\mathbf{R}, \boldsymbol{\kappa})$ **end if**
    counter $\leftarrow 0$, $(\mathbf{R}', \boldsymbol{\kappa}') \leftarrow (\mathbf{R}, \boldsymbol{\kappa})$ ▷ Perturbation phase
    while counter $< \chi$ **do**
        $(\mathbf{R}, \boldsymbol{\kappa}) \leftarrow$ Perturb Solution$((\mathbf{R}, \boldsymbol{\kappa}), \delta)$
        $(\mathbf{R}, \boldsymbol{\kappa}) \leftarrow$ Tabu Search$((\mathbf{R}, \boldsymbol{\kappa}), \nu, \zeta)$
        if $\bar{c}(\mathbf{R}, \boldsymbol{\kappa}) < \bar{c}\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right)$ **then** $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right) \leftarrow (\mathbf{R}, \boldsymbol{\kappa})$ **end if**
        if $\bar{c}(\mathbf{R}, \boldsymbol{\kappa}) < \bar{c}(\mathbf{R}', \boldsymbol{\kappa}')$ **then**
            counter $\leftarrow 0$
            $(\mathbf{R}', \boldsymbol{\kappa}') \leftarrow (\mathbf{R}, \boldsymbol{\kappa})$
        else
            counter $\leftarrow$ counter $+ 1$
        end if
    end while
end while
return $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right)$

customer is then selected from this list and inserted into the position that greedily minimizes the corresponding *insertion cost*. In our implementation, the restricted candidate list is cardinality-based and fixed to a pre-defined size η. The parameter η determines the extent of randomization and greediness during the construction process. Based on empirical evidence, a value of $\eta \in [1, 10]$ appeared to work well in our numerical experiments.

If a route was successfully constructed for at least one vehicle type, we select the route R of vehicle type k for which the *average cost per unit of carried load*, defined as $c(R, k)/\max_{q \in \mathcal{Q}} \sum_{i \in R} q_i$, is minimized. If no route could be constructed, then any remaining unrouted customer is inserted into an existing route (and corresponding position) for which a randomly weighted sum of the

capacity violation and insertion cost is minimized. We remark that efficient computation of the average cost per unit of carried load is enabled by the data structures described in Proposition 2.

Tabu search. In principle, we can replace all calls to the TABU SEARCH$((\mathbf{R}, \boldsymbol{\kappa}), \nu, \zeta)$ procedure by a simple local search algorithm that iteratively explores each pre-defined neighborhood to determine an improving solution. However, doing so will result in solutions that are only locally optimal with respect to the given neighborhoods, i.e., all neighbor solutions $(\mathbf{R}', \boldsymbol{\kappa}') \in \Omega_Y(\mathbf{R}, \boldsymbol{\kappa})$ in each of the pre-defined neighborhoods Y are non-improving: $\bar{c}(\mathbf{R}', \boldsymbol{\kappa}') \geq c(\mathbf{R}, \boldsymbol{\kappa})$. Tabu search [4] overcomes this shortcoming of local search and enhances its performance in two important ways: *(i)* non-improving moves are allowed, and *(ii)* improving moves may be disallowed. This enhancement is achieved using a short term memory (also known as a *tabu list*) to keep track of the most recently visited solutions in the search history and to prevent revisiting them for a predefined number of local search iterations ν (the *tabu tenure*). Any potential solution that has been visited within the last ν iterations is marked "tabu" (or forbidden) and inserted into the tabu list, so that the algorithm does not cycle by repeatedly visiting the same solutions. In fact, an admissible neighbor solution that is in the tabu list can be visited only if certain *aspiration criteria* are met; specifically, the tabu status of a solution is overridden only if it improves upon the best encountered solution. Moreover, the tabu search terminates if it performs ζ local search iterations without observing any further improvement. Typical values for these parameters are $\nu \in [20, 40]$ and $\zeta \in [100, 500]$ and in our implementation, we set $\nu = 30$ and $\zeta = 500$. Furthermore, we considered the set of neighborhoods to be the (intra- and inter-route) 1-0 relocate, 1-1 exchange and 2-opt neighborhoods. At each iteration, we randomly selected one of these neighborhoods and traversed it in lexicographic order, applying pruning mechanisms based on both feasibility and gain. The first improving neighbor solution replaced the current solution. Since the size of each of these neighborhoods is $\mathcal{O}(n^2)$, each iteration of tabu search itself can take $\mathcal{O}(n^2)$ time in the worst case, and its run time is largely dictated by the run time of the local search algorithm described in the previous section.

Perturbation mechanism. The procedure PERTURB SOLUTION$((\mathbf{R}, \boldsymbol{\kappa}), \delta)$ attempts to perturb the current solution $(\mathbf{R}, \boldsymbol{\kappa})$ such that the new solution cannot be encountered by application of tabu search alone. In our implementation, we consider a perturbation mechanism that first removes the route R of vehicle type k from the current solution which has the maximum value of *average cost*

per unit of carried load. In addition to this route, the mechanism also removes any routes R' that are "sufficiently close" to R, i.e., routes R' for which distance$(R, R') \coloneqq \max_{(i,j)\in R\times R'} \underline{c}_{ij} < \delta$. Here, $\underline{c}_{ij}$ is any suitably defined distance measure between customers i and j (e.g., geographical distance between i and j), and in our implementation, we set $\underline{c}_{ij} = \min_{k\in K} c_{ijk}$. If all routes R' satisfy distance$(R, R') \geq \delta$, then the route R'' which has the smallest distance to R is removed from the current solution. All customers that were visited on the deleted routes are then considered to be unrouted and are added back to the current partial solution using the same greedy insertion procedure that was used in the construction heuristic. We note that the parameter δ determines the extent of perturbation, with larger values of δ corresponding to higher extents of perturbation.

3 Hybrid Adaptive Memory Programming

Adaptive Memory Programming (AMP) is a metaheuristic that focuses on the exploitation of strategic memory components. Based on the intuition that high-quality locally optimal solutions share common features (e.g., common customer visiting sequences), AMP attempts to exploit a set of long-term memories (in contrast to the short-term memory used in tabu search) for the iterative construction of new *provisional solutions*. These solutions are used to restart and intensify the search, while adaptive learning mechanisms are applied to update the memory structures. We refer the reader to [4, 3] for a general overview of this subject. Our specific hybrid AMP implementation for the robust HVRP is described in Algorithm 2.

Algorithm 2 consists of two phases: an *initialization* phase (lines 2–7) and an *exploitation* phase (lines 8–13). The initialization phase populates the *reference set* $\mathcal{P}$ with μ solutions that are generated by first constructing an initial solution (line 3) and then improving it using tabu search (line 4). Once the initialization phase has completed, the exploitation phase manipulates $\mathcal{P}$ by exploring search trajectories initiated using the provisional solutions as starting points. Specifically, at each iteration of the exploitation phase, a provisional solution is first constructed by identifying common features of the reference solutions in $\mathcal{P}$ (line 9). This provisional solution is then further improved using tabu search (line 10) and inserted into the reference set $\mathcal{P}$ (line 12). The AMP algorithm terminates after a pre-specified time limit t_{lim} is reached (line 8), at which point the best encountered solution $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right)$ is returned (line 14). The procedures Construct Solution(η) and Tabu Search$(\left(\mathbf{R}, \boldsymbol{\kappa}\right), \nu, \zeta)$ along with their associated parameters η, ν and ζ are exactly the

Algorithm 2 Adaptive Memory Programming.

Input: μ, η, ν, ζ, and θ (user-defined parameters)

Start timer t, $\mathcal{P} \leftarrow \emptyset$, $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right) \leftarrow (\emptyset, \emptyset)$
while $|\mathcal{P}| < \mu$ **do** ▷ Initialization phase
    $(\mathbf{R}, \boldsymbol{\kappa}) \leftarrow$ Construct Solution(η)
    $(\mathbf{R}, \boldsymbol{\kappa}) \leftarrow$ Tabu Search$((\mathbf{R}, \boldsymbol{\kappa}), \nu, \zeta)$
    if $\bar{c}(\mathbf{R}, \boldsymbol{\kappa}) < \bar{c}\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right)$ **then** $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right) \leftarrow (\mathbf{R}, \boldsymbol{\kappa})$ **end if**
    $\mathcal{P} \leftarrow \mathcal{P} \cup (\mathbf{R}, \boldsymbol{\kappa})$
end while
while $t < t_{\text{lim}}$ **do** ▷ Exploitation phase
    $(\mathbf{R}, \boldsymbol{\kappa}) \leftarrow$ Construct Provisional Solution$(\mathcal{P}, \theta)$
    $(\mathbf{R}, \boldsymbol{\kappa}) \leftarrow$ Tabu Search$((\mathbf{R}, \boldsymbol{\kappa}), \nu, \zeta)$
    if $\bar{c}(\mathbf{R}, \boldsymbol{\kappa}) < \bar{c}\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right)$ **then** $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right) \leftarrow (\mathbf{R}, \boldsymbol{\kappa})$ **end if**
    $\mathcal{P} \leftarrow$ Update Reference Set$(\mathcal{P}, (\mathbf{R}, \boldsymbol{\kappa}))$
end while
return $\left(\mathbf{R}^B, \boldsymbol{\kappa}^B\right)$

same as in ILS (see Algorithm 1). The parameters θ and μ are used in the provisional construction and reference set update methods respectively, which we describe next.

Generation of provisional solutions. Provisional solutions are constructed by identifying and combining *elite components* from the reference set $\mathcal{P}$. We define an elite component to be a route associated with a particular vehicle type whose edges appear "sufficiently frequently" among the solutions in $\mathcal{P}$. The overall procedure Construct Provisional Solution$(\mathcal{P}, \theta)$ works as follows. We first attempt to generate a route for every vehicle type $k \in K$, assuming the number of routes of this type is less than m_k in the current solution. With probability θ, we construct a route using the members of $\mathcal{P}$, and with probability $1 - \theta$, we use the mechanism outlined in the basic Construct Solution(η) procedure. In the former case, we first assign a score to each route $R_h = (r_{h1}, \ldots, r_{h|R|})$ of vehicle type κ_h from the solution $(\mathbf{R}, \boldsymbol{\kappa}) \in \mathcal{P}$ as follows: $\text{score}(R_h, \kappa_h) = \text{w}(\mathbf{R}, \boldsymbol{\kappa}) \sum_{l=0}^{|R|} \text{freq}(r_{hl}, r_{hl+1}, k) \mathbb{1}_{[r_{hl}, r_{hl+1} \notin \text{current solution}]}$. Here $\text{w}(\mathbf{R}, \boldsymbol{\kappa})$ refers to the weight of solution $(\mathbf{R}, \boldsymbol{\kappa})$ while $\text{freq}(i, j, k)$ refers to the frequency with which edge $(i, j) \in E$ appears

among all routes of vehicle type k. Specifically, these quantities are defined as follows:

$$\mathrm{w}\left(\mathbf{R}, \boldsymbol{\kappa}\right) = \frac{\max_{(\mathbf{R}', \boldsymbol{\kappa}') \in \mathcal{P}} \bar{c}\left(\mathbf{R}', \boldsymbol{\kappa}'\right) - \bar{c}\left(\mathbf{R}, \boldsymbol{\kappa}\right)}{\max_{(\mathbf{R}', \boldsymbol{\kappa}') \in \mathcal{P}} \bar{c}\left(\mathbf{R}', \boldsymbol{\kappa}'\right) - \min_{(\mathbf{R}', \boldsymbol{\kappa}') \in \mathcal{P}} \bar{c}\left(\mathbf{R}', \boldsymbol{\kappa}'\right)},$$

$$\mathrm{freq}(i, j, k) = \sum_{(\mathbf{R}', \boldsymbol{\kappa}') \in \mathcal{P}} \sum_{h=1}^{H'} \mathbb{1}_{[\kappa'_h = k]} \sum_{l=0}^{|R'_h|} \mathbb{1}_{[(r'_{hl}, r'_{hl+1}) = (i,j)]}.$$

The route R with the highest score is then determined to be the candidate route of vehicle type k (after deleting those customers that have already been routed in the current solution).

Among all generated routes, the candidate route of the vehicle type with the lowest value of *average cost per unit of carried load* is then adopted in the current solution, and the entire procedure repeats. At the end, if unrouted customers still remain, then they are inserted into an existing route (and corresponding position) for which a randomly weighted sum of the capacity violation and insertion cost is minimized, similar to the basic construction heuristic. We recall that this is done so that the fleet availability constraints **(C2)** are never violated.

Reference set update method. In the initialization phase, the reference set $\mathcal{P}$ is grown to contain up to μ different solutions. In the exploitation phase, the size of $\mathcal{P}$ is kept constant by replacing older solutions with more recently encountered ones. To ensure an appropriate balance between quality and diversity among the reference solutions, the procedure Update Reference Set$(\mathcal{P}, (\mathbf{R}, \boldsymbol{\kappa}))$ uses a simple rule that replaces the worst solution (in terms of its total cost) $\left(\mathbf{R}^W, \boldsymbol{\kappa}^W\right)$ whenever the candidate solution to be inserted $(\mathbf{R}, \boldsymbol{\kappa})$ satisfies $\bar{c}\left(\mathbf{R}, \boldsymbol{\kappa}\right) < \bar{c}\left(\mathbf{R}^W, \boldsymbol{\kappa}^W\right)$.

4 Best Solutions Found

The following tables report the best solutions found for each of the benchmark instances and uncertainty sets that we have considered in the paper. The budget sets, factor models and general ellipsoids correspond to the setting of $(\alpha, \beta) = (0.1, 0.5)$, the cardinality-constrained and discrete sets correspond to $(\alpha, \beta) = (0.1, 0.2)$, while the axis-parallel ellipsoid corresponds to $(\alpha, \beta) = (0.1, 1.0)$. In each table, "Inst" denotes the instance numbered as per the original dataset, n and m denote the number of customers and vehicle types respectively, while the quantity reported under uncertainty set $\mathcal{Q}$ is the objective value of the best found solution for that instance and setting of the uncertainty set.

Table 2. Best solutions found for the HVRPFD instances.

Inst	n	m	$\{q^0\}$	$\mathcal{Q}_B$	$\mathcal{Q}_F$	$\mathcal{Q}_E^{\text{ax}}$	$\mathcal{Q}_E^{\text{gen}}$	$\mathcal{Q}_G$	$\mathcal{Q}_S$
13	50	6	2,929.54	3,183.32	3,136.73	3,061.73	3,093.55	3,185.09	3,047.35
14	50	3	9,584.67	10,106.67	10,103.02	9,600.38	9,605.91	10,106.67	9,599.48
15	50	3	2,761.41	3,065.29	2,965.21	2,941.70	2,941.70	3,065.29	2,934.85
16	50	3	3,085.06	3,265.41	3,238.55	3,145.47	3,221.38	3,265.41	3,134.01
17	75	4	1,960.59	2,076.96	2,067.28	2,001.71	2,013.99	2,076.96	1,991.14
18	75	6	3,524.34	3,748.68	3,709.86	3,628.33	3,662.54	3,745.10	3,618.75
19	100	3	9,693.23	10,420.34	10,420.34	9,701.73	9,748.98	10,420.34	9,701.64
20	100	3	4,469.86	4,795.14	4,731.65	4,599.16	4,714.54	4,834.17	4,612.88

Table 3. Best solutions found for the HVRPD instances.

Inst	n	m	$\{q^0\}$	$\mathcal{Q}_B$	$\mathcal{Q}_F$	$\mathcal{Q}_E^{\text{ax}}$	$\mathcal{Q}_E^{\text{gen}}$	$\mathcal{Q}_G$	$\mathcal{Q}_S$
13	50	6	1,432.57	1,517.84	1,502.24	1,481.13	1,482.49	1,517.84	1,468.83
14	50	3	584.38	606.67	603.02	596.53	597.54	606.67	596.63
15	50	3	961.41	1,015.29	1,012.64	991.70	991.70	1,015.29	984.85
16	50	3	1,085.06	1,144.94	1,133.16	1,120.57	1,121.38	1,144.94	1,110.41
17	75	4	1,009.48	1,061.96	1,052.28	1,021.50	1,023.99	1,061.96	1,020.28
18	75	6	1,761.88	1,823.58	1,812.32	1,783.93	1,788.38	1,823.58	1,777.02
19	100	3	1,093.23	1,120.34	1,120.34	1,101.64	1,116.47	1,120.34	1,101.64
20	100	3	1,472.97	1,534.17	1,533.62	1,501.53	1,514.39	1,534.17	1,508.31

Table 4. Best solutions found for the FSMFD instances.

Inst	n	m	$\{q^0\}$	$\mathcal{Q}_B$	$\mathcal{Q}_F$	$\mathcal{Q}_E^{\text{ax}}$	$\mathcal{Q}_E^{\text{gen}}$	$\mathcal{Q}_G$	$\mathcal{Q}_S$
3	20	5	986.93	1,144.22	1,118.76	1,106.84	1,106.65	1,144.22	1,092.59
4	20	3	6,377.28	6,437.33	6,432.25	6,413.06	6,413.06	6,437.33	6,392.34
5	20	5	1,167.40	1,322.26	1,298.72	1,287.48	1,287.48	1,322.26	1,249.08
6	20	3	6,416.11	6,516.47	6,500.82	6,499.74	6,499.74	6,516.47	6,470.82
13	50	6	2,711.61	2,964.65	2,935.40	2,906.68	2,873.24	2,964.65	2,908.96
14	50	3	8,582.05	9,126.90	8,644.00	8,618.16	8,618.16	9,126.90	8,618.16
15	50	3	2,450.82	2,634.96	2,608.61	2,590.47	2,591.86	2,634.96	2,591.93
16	50	3	2,906.71	3,168.92	3,137.50	3,061.09	3,070.34	3,168.92	3,052.39
17	75	4	1,872.49	2,004.48	1,969.64	1,932.82	1,946.09	2,004.48	1,925.47
18	75	6	2,918.45	3,153.09	3,100.08	3,056.88	3,064.50	3,152.16	3,063.70
19	100	3	8,094.97	8,662.86	8,649.99	8,132.14	8,409.14	8,662.86	8,134.19
20	100	3	3,839.11	4,165.91	4,138.25	4,046.05	4,085.22	4,168.44	4,054.18

Table 5. Best solutions found for the FSMF instances.

Inst	n	m	$\{q^0\}$	$\mathcal{Q}_B$	$\mathcal{Q}_F$	$\mathcal{Q}_E^{\text{ax}}$	$\mathcal{Q}_E^{\text{gen}}$	$\mathcal{Q}_G$	$\mathcal{Q}_S$
3	20	5	887.99	954.37	949.79	921.23	927.96	951.61	918.63
4	20	3	6,327.60	6,437.33	6,432.25	6,413.06	6,413.06	6,437.33	6,379.27
5	20	5	919.86	1,005.27	980.57	961.63	963.63	988.63	949.85
6	20	3	6,353.72	6,516.47	6,500.82	6,499.74	6,499.74	6,516.47	6,448.52
13	50	6	2,223.70	2,399.20	2,365.21	2,305.17	2,327.57	2,406.36	2,304.11
14	50	3	8,562.41	9,119.03	8,644.00	8,618.16	8,618.16	9,119.03	8,618.16
15	50	3	2,360.49	2,586.37	2,561.65	2,475.63	2,492.99	2,586.37	2,463.19
16	50	3	2,506.66	2,721.40	2,697.88	2,610.03	2,646.51	2,720.43	2,612.59
17	75	4	1,621.25	1,734.53	1,712.24	1,658.29	1,680.08	1,734.53	1,657.43
18	75	6	2,195.13	2,369.65	2,322.62	2,272.55	2,294.51	2,369.65	2,267.88
19	100	3	8,094.97	8,662.86	8,649.99	8,129.33	8,382.79	8,662.86	8,135.02
20	100	3	3,714.44	4,043.97	3,973.46	3,862.99	3,911.89	4,060.73	3,874.70

Table 6. Best solutions found for the FSMD instances.

Inst	n	m	$\{q^0\}$	$\mathcal{Q}_B$	$\mathcal{Q}_F$	$\mathcal{Q}_E^{\text{ax}}$	$\mathcal{Q}_E^{\text{gen}}$	$\mathcal{Q}_G$	$\mathcal{Q}_S$
3	20	5	555.66	623.22	609.51	604.02	604.02	623.22	585.02
4	20	3	369.77	378.70	373.24	369.77	373.24	380.71	373.24
5	20	5	713.62	742.87	737.36	736.90	736.90	742.85	721.00
6	20	3	391.42	414.66	415.03	396.26	405.46	406.19	396.26
13	50	6	1,405.87	1,491.86	1,488.78	1,468.28	1,471.82	1,491.86	1,460.06
14	50	3	575.63	603.21	601.71	583.94	588.64	603.21	583.94
15	50	3	944.96	999.82	997.98	979.40	985.19	999.82	974.59
16	50	3	1,078.67	1,131.00	1,114.78	1,110.88	1,111.69	1,131.00	1,107.74
17	75	4	1,006.50	1,036.54	1,029.75	1,017.52	1,020.50	1,038.60	1,016.64
18	75	6	1,746.53	1,800.80	1,795.02	1,778.61	1,785.58	1,800.80	1,775.35
19	100	3	1,073.71	1,106.17	1,101.07	1,084.54	1,099.69	1,105.44	1,093.74
20	100	3	1,468.79	1,530.52	1,525.33	1,495.31	1,514.46	1,533.24	1,497.01

Table 7. Best solutions found for the SDVRP instances.

Inst	n	m	$\{q^0\}$	$\mathcal{Q}_B$	$\mathcal{Q}_F$	$\mathcal{Q}_E^{\text{ax}}$	$\mathcal{Q}_E^{\text{gen}}$	$\mathcal{Q}_G$	$\mathcal{Q}_S$
1	50	3	609.52	640.32	634.19	621.50	624.77	640.32	623.37
2	50	2	563.85	598.10	593.59	569.45	577.70	598.10	569.45
3	75	3	899.90	954.32	938.11	908.95	922.33	954.32	908.95
4	75	2	806.72	854.43	843.73	831.60	836.25	854.43	825.27
5	100	3	963.09	1,003.57	993.64	977.14	978.00	1,003.57	976.01
6	100	2	965.86	1,028.52	1,016.95	987.88	1,000.59	1,028.52	983.46
7	27	3	391.30	391.30	391.30	391.30	391.30	391.30	391.30
8	54	3	664.46	664.46	664.46	664.46	664.46	664.46	664.46
9	81	3	948.23	948.23	948.23	948.23	948.23	948.23	948.23
10	108	3	1,189.69	1,218.75	1,218.75	1,208.74	1,208.74	1,218.75	1,200.34
11	135	3	1,411.03	1,449.63	1,448.17	1,440.49	1,453.43	1,449.63	1,439.59
13	54	3	1,150.31	1,194.18	1,194.18	1,162.85	1,171.58	1,194.18	1,171.58
14	108	3	1,873.74	1,960.88	1,960.88	1,886.53	1,908.76	1,960.88	1,889.59
19	100	3	816.74	843.15	836.79	822.92	831.11	844.19	827.58
20	150	3	993.42	1,034.63	1,026.28	1,006.06	1,032.96	1,041.07	1,014.66
22	120	3	977.54	1,008.71	1,001.98	983.72	991.86	1,008.71	983.96
23	100	3	764.19	803.29	803.29	781.81	784.83	803.29	781.81

Table 8. Best solutions found for the MDVRP instances.

Inst	n	m	$\{q^0\}$	$\mathcal{Q}_B$	$\mathcal{Q}_F$	$\mathcal{Q}_E^{\text{ax}}$	$\mathcal{Q}_E^{\text{gen}}$	$\mathcal{Q}_G$	$\mathcal{Q}_S$
1	50	4	560.63	576.87	570.81	570.81	570.81	576.87	575.36
2	50	4	464.35	473.53	473.01	464.48	464.48	473.22	464.48
3	75	5	623.79	640.65	635.93	629.86	630.98	641.19	629.67
4	100	2	951.28	999.21	997.20	973.68	982.38	999.21	978.74
5	100	2	725.55	750.03	747.29	731.27	741.44	750.03	732.17
6	100	3	838.00	876.50	871.47	864.49	867.39	877.26	865.93
7	100	4	843.30	881.97	871.31	855.39	860.65	881.97	855.55
12	80	2	1,290.93	1,318.95	1,314.36	1,314.36	1,314.36	1,318.95	1,301.68
15	160	4	2,463.14	2,505.42	2,505.42	2,505.42	2,505.42	2,505.42	2,492.55

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[2] George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5:266–277, 1957.

[3] Éric D Taillard, Luca M Gambardella, Michel Gendreau, and Jean-Yves Potvin. Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1 – 16, 2001.

[4] Fred Glover. Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges. In Richard S. Barr, Richard V. Helgason, and Jeffery L. Kennington, editors, *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, pages 1–75. Springer US, Boston, MA, 1997.

[5] Chrysanthos E. Gounaris, Wolfram Wiesemann, and Christodoulos A. Floudas. The Robust Capacitated Vehicle Routing Problem Under Demand Uncertainty. *Operations Research*, 61(3):677–693, 2013.

[6] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 320–353. Springer US, Boston, MA, 2003.