Annals of Operations Research (2024) 343:1–38
https://doi.org/10.1007/s10479-024-06295-9

ORIGINAL RESEARCH

# CMSA based on set covering models for packing and routing problems

**Mehmet Anıl Akbay**[1] · **Christian Blum**[1] · **Can Berk Kalayci**[2]

Received: 24 September 2023 / Accepted: 13 September 2024 / Published online: 8 October 2024


**Abstract**
Many packing, routing, and knapsack problems can be expressed in terms of integer linear programming models based on set covering. These models have been exploited in a range of successful heuristics and exact techniques for tackling such problems. In this paper, we show that integer linear programming models based on set covering can be very useful for their use within an algorithm called "Construct, Merge, Solve & Adapt"(CMSA), which is a recent hybrid metaheuristic for solving combinatorial optimization problems. This is because most existing applications of CMSA are characterized by the use of an integer programming solver for solving reduced problem instances at each iteration. We present applications of CMSA to the variable-sized bin packing problem and to the electric vehicle routing problem with time windows and simultaneous pickups and deliveries. In both applications, CMSA based on a set covering model strongly outperforms CMSA when using an assignment-type model. Moreover, state-of-the-art results are obtained for both considered optimization problems.

**Keywords** Construct, Merge, Solve & Adapt · Set covering models · Bin packing · Routing

## 1 Introduction

Construct, Merge, Solve & Adapt (CMSA) (Blum et al., 2016) is a hybrid metaheuristic for solving combinatorial optimization problems. It is an iterative approach based on solving a subinstance of the considered problem instance at each iteration. In this context, note that the search space (set of feasible solutions) of a sub-instance is a subset of the search space of the original problem instance. In other words, every feasible solution to a subinstance is also a feasible solution to the original problem instance. Today's existing CMSA variants

Christian Blum and Can Berk Kalayci have contributed equally to this work.

✉ Mehmet Anıl Akbay
makbay@iiia.csic.es

Christian Blum
christian.blum@iiia.csic.es

Can Berk Kalayci
cbkalayci@pau.edu.tr

1 Artificial Intelligence Research Institute (IIIA-CSIC), Campus of the UAB, 08193 Bellaterra, Spain

2 Department of Industrial Engineering, Pamukkale University, Denizli, Turkey

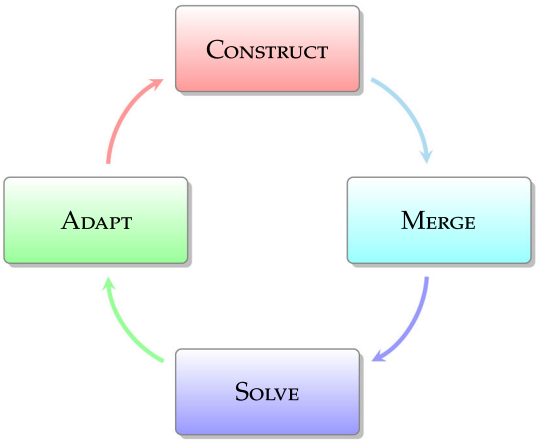



**Fig. 1** General framework of any CMSA approach

are all covered by the algorithmic framework shown in Fig. 1. After generating an initial subinstance—which is empty in the simplest case—any CMSA algorithm executes four different steps at each algorithm iteration. First, in the Construct step, valid solutions to the considered problem instance are generated in some way. Then, in the Merge step, the solution components found in these solutions are added to the incumbent subinstance, which is hereby extended. The incumbent subinstance is solved in the third step, the Solve step. Note, in this context, that it is not necessarily the case that the subinstance is solved to optimality. Finally, an aging mechanism is employed in the Adapt step to adapt the subinstance based on the outcome of the Solve step. This adaptation reduces the incumbent subinstance by removing some of the solution components identified by the aging mechanism.

While the Merge and Adapt steps are implemented in the same way in today's existing CMSA variants, the Construct and Solve steps offer a rather large degree of freedom. In the case of the Construct step, valid solutions to the considered problem instance might, for example, be generated by using a probabilistic greedy heuristic. A second option consists of feeding the incumbent subinstance of CMSA with solution components from solutions generated by a metaheuristic approach run in parallel, for example. Concerning the Solve step, an integer linear programming (ILP) solver might be used to solve the incumbent subinstance if the considered optimization problem can be expressed by means of an ILP model. Other options include using specialized exact techniques and short runs of an available metaheuristic for the tackled problem.

In this paper, we study different ways of solving the incumbent subinstance in the Solve step of the CMSA framework. We do so in the context of the application of CMSA to combinatorial optimization problems whose goal is the partitioning of a finite set of items into disjoint subsets. This problem type comprises large families of important problems such as bin packing problems, multiple knapsack problems, assembly line balancing problems, and vehicle routing problems, just to mention a few. ILP solvers such as CPLEX and Gurobi find it generally difficult to solve standard assignment-type ILP models of these problems. Therefore, the Operations Research community has developed specialized exact techniques based on set covering models for solving them. Concerning exact techniques, set covering models are especially useful in the context of column generation methods; see, for example, (Barnhart et al., 1998; Desrochers & Soumis, 1989; Parker & Ryan, 1993). However, the transformation of vehicle routing and packing problems to set covering problems has

also been exploited in the context of heuristic methods; see, for example, (Monaci & Toth, 2006; Cacchiani et al., 2014; Machado et al., 2021). In this paper, we show that replacing the standard assignment-type ILP models with set covering formulations for solving subinstances within CMSA algorithms may often result in high-performance techniques. In addition, these set-covering-based, high-performance CMSA techniques are, in contrast to column generation techniques, rather easy to program.

### 1.1 Literature Review

As mentioned above, different CMSA variants—that is, different instantiations of the framework shown in Fig. 1—have been proposed in the related literature. The most-utilized CMSA variant to date is a generic CMSA based on probabilistically constructing solutions in the CONSTRUCT step and using an ILP solver for solving subinstances in the SOLVE step. First, this algorithm has been applied to the minimum covering arborescence problem and the minimum common string partition problem (Blum et al., 2016). More recent applications include the ones to prioritized pairwise test data generation in software product lines (Ferrer et al., 2021), the maximum happy vertices problem (Thiruvady & Lewis, 2022), and the max tree coverage problem (Zhou & Zhang, 2024). A more recent CMSA variant is self-adaptive CMSA, proposed in Akbay et al. (2022). This CMSA variant is characterized by the fact that the values of important algorithm parameters are handled in a self-adaptive way and that the probabilistic generation of valid solutions in the CONSTRUCT step is biased towards the best-found solution. Existing applications of this CMSA variant include the one to the multi-dimensional multi-way number partitioning (Djukanović et al., 2023) and the two-echelon electric vehicle routing problem with simultaneous pickup and deliveries (Akbay et al., 2023). Finally, it is worth mentioning that CMSA has already been applied to complex, practical problems such as the optimization of refueling and maintenance planning of nuclear power plants (Dupin & Talbi, 2021).

Next, we review works from the literature based on ideas similar to CMSA's. The core concept of CMSA closely aligns with that found in large neighborhood search (LNS) (Pisinger & Røpke, 2010). Like CMSA, LNS is based on iteratively solving subinstances of the tackled problem instance. The main difference is in the way in which these subinstances are generated and adapted from one iteration to the next. Applegate et al. (1999) and Cook and Seymour (2003) solved the classical traveling salesman problem (TSP) as follows. Initially, in a preliminary phase, they employ a metaheuristic to generate a collection of high-quality TSP solutions. Subsequently, these solutions are merged to create a subinstance, which is then solved by an exact solver. In other words, their algorithm applies one iteration of CMSA (without the execution of the ADAPT step). A similar approach is used in (Cavaliere et al., 2022) for the capacitated vehicle routing problem (CVRP). Another example of work related to CMSA can be found in (Nepomuceno et al., 2007), where the so-called *generate-and-solve* (GS) framework is proposed. A recent application of this framework is presented in (Sá Santos & Nepomuceno, 2022). The method known as *kernel search* (Angelelli et al., 2010) employs a heuristic strategy that involves identifying a limited set of solution components likely to be present in optimal solutions (the kernel) and then utilizing an ILP solver to find the best solutions that contain the kernel; see (Lamanna et al., 2022) for a recent application. Another approach related to CMSA worth mentioning is *merge search* (MS) (Kenny et al., 2018). Like CMSA, MS creates a subinstance in each iteration and attempts to solve it with an exact solver. However, the key difference lies in how these subinstances are formed. CMSA focuses on identifying a set of variables with fixed values in high-quality solutions, shifting

optimization to the remaining variables. In contrast, MS groups variables with consistent values across good solutions, although the exact values within these groups are still optimized. An example of MS application can be found in Thiruvady et al. (2020).

### 1.2 Our Contribution

The contribution of this work is twofold. First, we provide a conceptual contribution and show that—in the context of problems that can be modeled both by means of standard assignment-type ILP models and by set-covering-based ILP models—the use of set-covering-based models for solving subinstances within CMSA may significantly outperform the use of standard assignment-type models. This is shown both for a problem from the bin packing field and for a complex problem from the field of electric vehicle routing. The paper's second contribution is to be found in the state-of-the-art results that our algorithms obtain for both considered problems. In fact, in the case of the variable-sized bin packing problem, new best-known solutions are found in 68 out of 150 cases. Furthermore, in the context of the electric vehicle routing problem, we compare to our previous work in which this complex problem was introduced and we are able to show that the results obtained by our new CMSA variant significantly outperform previous results.

## 2 CMSA: construct, merge, solve & adapt

In this section, we describe the generic CMSA algorithm variant, which will be used in our first application example concerning the variable-sized bin packing problem. The first action that needs to be taken for applying CMSA to a combinatorial optimization problem is defining the set $C$ of solution components, that is, those components of which solutions to the considered problem are composed. Later we will provide specific examples for such a definition. For the moment, however, let $C = \{c_1, \ldots, c_n\}$ be a generic set of solution components. Moreover, any valid solution $S \in \mathscr{S}$ to our problem—where $\mathscr{S}$ is the set of all valid solutions—can be expressed as a subset of $C$, that is, $S \subseteq C$ for all $S \in \mathscr{S}$. Finally, let $f : \mathscr{S} \mapsto \mathbb{N}^+$ for the following discussion be the objective function to be minimized, and let $f(\emptyset) := \infty$.

Algorithm 1 provides the pseudo-code of our generic CMSA. The algorithm starts by initializing both the best-so-far solution $S^{\text{bsf}}$ and the subinstance $C'$, which is always a subset of $C$, to the empty set. Furthermore, the so-called age values of all solution components are initialized to zero, that is, $age[c] := 0$ for all $c \in C$. The main loop of the CMSA algorithm consists of four actions, as already shown in the general CMSA framework from Fig. 1. First, in the CONSTRUCT step of CMSA, a number of $n_a$ valid solutions to the considered problem are probabilistically generated (see line 8 of Algorithm 1). Second, in the MERGE step of CMSA, the current subinstance $C'$ is updated with the solution components found in these $n_a$ solutions (see lines 10–13). Third, in the SOLVE step of CMSA, an ILP solver is applied in order to find the best possible solution that only contains components from subinstance $C'$, within a time limit of $t_{\text{ILP}}$ CPU seconds (see line 15). Fourth, in the ADAPT step of CMSA, subinstance $C'$ is adapted based on the solution $S^{\text{ILP}}$ returned by the ILP solver (see line 17). These four steps are iterated until a given CPU time limit is reached. The output of the algorithm is $S^{\text{bsf}}$, the best solution found during the whole process.

Note that the CONSTRUCT step and the SOLVE step of CMSA are problem-dependent. In particular, for the CONSTRUCT step of the algorithm, generally, a greedy heuristic for the

**Algorithm 1** Pseudo-code of a generic CMSA

**input 1:** Complete set of solution components $C$
**input 2:** values for CMSA parameters $n_a$, $age_{max}$, and $t_{ILP}$
$S^{bsf} := \emptyset$
$C' := \emptyset$
$age[c] := 0$ for all $c \in C$
**while** CPU time limit not reached **do**
  **for** $i := 1, \ldots, n_a$ **do**
    $S :=$ ProbabilisticSolutionConstruction($C$)
    **if** $f(S) < f(S^{bsf})$ **then** $S^{bsf} := S$ **endif**
    **for** all $c \in S$ **and** $c \notin C'$ **do**
      $age[c] := 0$
      $C' := C' \cup \{c\}$
    **end for**
  **end for**
  $S^{ILP} :=$ SolveSubinstance($C', t_{ILP}$)
  **if** $f(S^{ILP}) < f(S^{bsf})$ **then** $S^{bsf} := S^{ILP}$ **end if**
  Adapt($C', S^{ILP}, age_{max}$)
**end while**
**output:** $S^{bsf}$

tackled problem is used in a randomized way, and the SOLVE step depends on the availability of an ILP model for the tackled problem. Moreover, the way in which an ILP model is exactly generated based on subinstance $C'$ leaves room for variation. In contrast, the MERGE step and the ADAPT step are problem-independent. In the MERGE step, those solution components that (1) are found in at least one of the $n_a$ constructed solutions and (2) do currently not form part of $C'$, are added to $C'$ and their age value is set to zero. Finally, the ADAPT step consists in the application of function Adapt($C', S^{ILP}, age_{max}$), which adapts subinstance $C'$ in the following way. First, the age values of all components in $C' \setminus S^{ILP}$ are incremented by one. Second, the age values of all components in $S^{ILP}$ are set to zero. The final action in the ADAPT step removes all those components from $C'$ whose age value has reached the maximum allowed age ($age_{max}$). This is done in order to prevent components that never appear in $S^{ILP}$ from slowing down the ILP solver in subsequent CMSA iterations. Note that the age value $age[c]$ of a solution component $c \in C$, at any time, indicates the number of consecutive CMSA iterations for which $c$ has formed part of subinstance $C'$ without having been included in the ILP-solution to the subinstance $C'$.

## 3 Application to variable-sized bin packing

The variable-sized bin packing problem (VSBPP) can technically be described as follows. Given is a set $S = \{1, \ldots, n\}$ of $n$ items, and each item $i \in S$ has a weight $w_i > 0$. Also given is a set $B = \{1, \ldots, m\}$ of $m$ bin types. Hereby, a bin type $k \in B$ is characterized by a capacity $W_k > 0$ and a cost $C_k$. We henceforth assume, without loss of generality, that $W_1 < \ldots < W_m$. The goal of the VSBPP is to pack the $n$ items into a number of bins such that sum of the costs of the utilized bins is minimized. Note that there is no restriction on the number of times a bin type may be used.

### 3.1 Assignment-Type ILP Model of the VSBPP

The VSBPP can be modeled in the following way as an assignment-type integer linear program; see also (Haouari & Serairi, 2009). This model is henceforth denoted by $\texttt{ILP}_{\text{std}}^{\text{VSBPP}}$:

$$\textbf{min} \quad \sum_{j=1}^{n} \sum_{k=1}^{m} C_k \cdot y_{jk} \tag{1}$$

$$\textbf{s.t.} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \text{for } i = 1, \ldots, n \tag{2}$$

$$\sum_{k=1}^{m} y_{jk} \leq 1 \quad \text{for } j = 1, \ldots, n \tag{3}$$

$$\sum_{i=1}^{n} w_i \cdot x_{ij} \leq \sum_{k=1}^{m} W_k \cdot y_{jk} \quad \text{for } j = 1, \ldots, n \tag{4}$$

$$x_{ij} \in \{0, 1\} \quad \text{for } i, j = 1, \ldots, n \tag{5}$$

$$y_{jk} \in \{0, 1\} \quad \text{for } j = 1, \ldots, n \text{ and } k = 1, \ldots, m \tag{6}$$

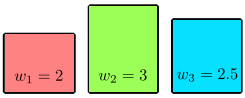
This ILP model makes use of two sets of binary variables. A setting of $x_{ij} = 1$, for example, means that item $i$ is placed in bin $j$. Moreover, a setting of $y_{jk} = 1$ means that bin $j$ has bin type $k$. Note also that the number of used bins can safely be limited to $n$, which is the number of items. Constraints (2) make sure that each item is assigned to exactly one bin. In a similar way, constraints (3) enforce that each used bin has exactly one bin type. In addition, constraints (4) ensure that bin capacities are respected. Finally, note that the VSBPP is NP-hard due to being a generalization of the one-dimensional bin packing problem. For an example of a small VSBPP problem instance, see Fig. 2.

### 3.2 Literature Review Concerning the VSBPP

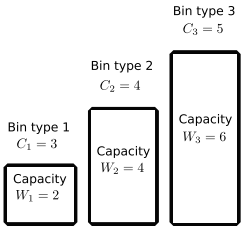
We focus on the original version of the VSBPP, where—as mentioned above—the number of bins available per bin type is unlimited. Crainic et al. (2011) derived lower bounds and developed heuristics for the more general version of the VSBPP with explicit limits on the number of bins per bin type. Haouari and Serairi (2009) introduced a range of greedy heuristics, in addition to a genetic algorithm. Finally, Hemmelmayr et al. (2012) proposed a rather sophisticated variable neighborhood search (VNS) algorithm for solving the VSBPP. The results presented in (Hemmelmayr et al., 2012) have not improved since then. Instead, researchers have focused on the VSBPP with additional constraints. Recent examples include the VSBPP with time windows (Fleszar, 2023) and the VSBPP with conflicts (Ekici, 2023).

### 3.3 Set-Covering Based ILP Model of the VSBPP

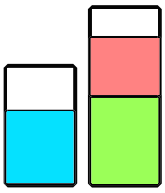
An alternative, set-covering-based ILP model for the VSBPP is obtained as follows. Let $\mathscr{B}$ be the set of all possible bins with respect to their content. Hereby, a bin $b \in \mathscr{B}$ is defined by the set of items it contains. The weight $w_b$ of a bin $b \in \mathscr{B}$ is defined as the sum of the weights of the items assigned to that bin. Moreover, the cost $c_b$ of a bin $b \in \mathscr{B}$ is defined as the cost of the lowest-cost bin type possible for accommodating all the items assigned to $b$. Finally, let $\mathscr{B}_i \subset \mathscr{B}$ be the set of bins that contain item $i$. After introducing a binary variable



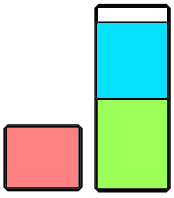

(a) Three items




(b) Three bin types



(c) A valid solution with cost $4 + 5 = 9$.



(d) Optimal solution with cost $3 + 5 = 8$.

**Fig. 2** An example instance with $n = 3$ items and $m = 3$ different bin types. The item weights are provided in (**a**) while the bin type capacities and costs are indicated in (**b**). **c** shows a valid solution in which item 3 is assigned to a bin of type 2 (and cost 4), while items 1 and 2 are assigned to a bin of type 3 (and cost 5). Therefore, the solution in (**c**) has value $4 + 5 = 9$. (**d**) shows the optimal solution where item 1 is assigned to a bin of type 1 (and cost 3), while items 2 and 3 are assigned to a bin of type 3 (and cost 5). The cost of the optimal solution is, therefore, $3 + 5 = 8$

$x_b$ for each bin $b \in \mathscr{B}$, the set-covering-based ILP model for the VSBPP, henceforth denoted as $\texttt{ILP}^{\text{VSBPP}}_{\text{setcov}}$, can be stated as follows.

$$\textbf{min} \sum_{b \in \mathscr{B}} c_b \cdot x_b \tag{7}$$

$$\textbf{s.t.} \quad \sum_{b \in \mathscr{B}_i} x_b \geq 1 \quad \text{for } i = 1, \ldots, n \tag{8}$$

$$x_b \in \{0, 1\} \quad \text{for all } b \in \mathscr{B} \tag{9}$$

Note that an exact correspondence between $\texttt{ILP}^{\text{VSBPP}}_{\text{std}}$ and $\texttt{ILP}^{\text{VSBPP}}_{\text{setcov}}$ would be obtained by replacing the"$\geq$"symbol in constraints (8) by the equality symbol ("="). However, every optimal solution obtained with the"$\geq$" symbol is easily transformed into an optimal solution of the model with the"="symbol by removing duplicate items from all bins apart from one. Moreover, according to Barnhart et al. (1998), the linear programming relaxation of the model when using the "$\geq$"symbol is numerically far more stable and thus easier to solve, which facilitates solving the ILP by means of ILP solvers such as CPLEX or Gurobi.

### 3.4 Application of standard CMSA to the VBSPP

First, we applied CMSA in the following standard way to the VBSPP. This algorithm variant is henceforth labeled Adapt-CMSA- STD. As ILP model for solving subinstances, Adapt-CMSA- STD uses model $\texttt{ILP}^{\text{VSBPP}}_{\text{std}}$ from Sect. 3.1. A generic way of defining the

set of solution components is as follows. For each binary variable of the model, exactly two solution components are introduced: one component that corresponds to setting the variable to zero, and another component that refers to setting the variable to one. In the case of model $\texttt{ILP}_{\text{std}}^{\text{VSBPP}}$, this means that $C$ consists of solution components $cx_{ij}^0$ and $cx_{ij}^1$ for all binary variables $x_{ij}$ $(i, j, = 1, \ldots, n)$, and of solution components $cy_{jk}^0$ and $cy_{jk}^1$ for all binary variables $y_{jk}$ $(j = 1, \ldots, n;\ k = 1, \ldots, m)$. Hereby, $cx_{ij}^0$, for example, corresponds to $x_{ij} = 0$, while $cx_{ij}^1$ corresponds to $x_{ij} = 1$. Moreover, $C = \{cx_{11}^0, \ldots, cx_{nn}^0, cx_{11}^1, \ldots, cx_{nn}^1, cy_{11}^0, \ldots, cy_{nm}^0, cy_{11}^1, \ldots, cy_{nm}^1\}$ is the complete set of $2n^2 + 2nm$ solution components. Any valid solution $S$ is a subset of $C$ with $|S| = n^2 + nm$ because, for each binary variable, a solution $S$ contains exactly one of the two corresponding solution components.

### 3.4.1 Probabilistic Solution Construction

For the following discussion, a bin $b$ is a set of items, that is, $b \subseteq \{1, \ldots, n\}$. Moreover, a bin $b$ is always characterized by three well-defined measures:

1. $b_{\text{load}}$: the load of a bin $b$ is defined as the sum of the weights of the items it contains, that is, $b_{\text{load}} := \sum_{i \in b} w_i$
2. $b_{\text{type}}$: the type of a bin $b$ is defined as the lowest-cost bin type that is able to accommodate the load of the bin, that is, $b_{\text{type}} := k$ such that $C_k < C_r$ for all $r \in \{1, \ldots, m\}$ with $W_r \geq b_{\text{load}}$.
3. $b_{\text{cost}}$: the cost of a bin is defined as the cost of its type, that is, $b_{\text{cost}} := C_{b_{\text{type}}}$.
4. $b_{\text{ratio}}$: the ratio between the cost and the load of a bin, that is, $b_{\text{ratio}} := \frac{b_{\text{cost}}}{b_{\text{load}}}$

In addition, let $\max_{\text{load}}$ be defined as the maximum capacity of all bin types, that is, $\max_{\text{load}} := \max\{W_j \mid j = 1, \ldots, m\}$. For the probabilistic construction of a solution, the following simple procedure is applied; see also Algorithm 2. First, the $n$ items are randomly ordered; see line 3. Then, the set of bins $B$ is initialized by placing the first item from the list in a new bin, whose load, type, cost and ratio is determined as defined above. Then, in the pre-determined order, the remaining items are placed into bins. In particular, in probability, among all options to place an item, the one resulting in a bin with a lower ratio is preferred over the others. This is encapsulated in function $\mathsf{ChooseOption}(O, d_{\text{rate}}, l_{\text{size}})$ (see line 16), where $O$ is the current set of options. The working of this function is as follows. First, a number $z$ is chosen uniformly at random from [0, 1]. In case $z \leq d_{\text{rate}}$, the chosen option is the one with the lowest ratio. Otherwise, the $\max\{|O|, l_{\text{size}}\}$ options with the lowest ratios are pre-selected from $O$, and the chosen option is randomly determined among those. When all items are placed into bins, the set of bins is sorted by bin ratio (from small to large); see function $\mathsf{Sort}(B)$ in line 22. As a tie-breaking criterion, we utilized the smallest item index of a bin (preferring smaller ones). Finally, the constructed solution is transformed into the corresponding set $S$ of solution components in function $\mathsf{ExtractSolutionComponents}(B)$; line 23. In the case of the set of solution components outlined above, this works as follows. If the first bin (after sorting $B$) is of type $k$, then $cy_{1k}^1$ is added to $S$. Moreover, all $cy_{1r}^0$ (with $r \neq k \in \{1, \ldots, m\}$) are added to $S$. The same is done for all other bins in $B$. Similarly, for each item $i$ in the first bin (after sorting $B$), component $cx_{i1}^1$ is added to $S$. Moreover, all $cx_{ir}^0$ (with $r = 2, \ldots, n$) are added to $S$. The same is done for the items of all other bins from $B$.

**Algorithm 2** Probabilistic construction of a valid VSBPP solution

**input:** values for solution construction parameters $d_{\text{rate}}, l_{\text{size}}$
Let $(i_1, \ldots, i_n)$ be a randomly ordered list of all $n$ items
$b^{\text{init}} := \{i_1\}$ *#Comment: $b^{\text{init}}$ is the first bin that is initialized*
$B := \{b^{\text{init}}\}$
**for** $l := 2, \ldots, n$ **do**
  $i := i_l$
  $O := \emptyset$
  **for** $b \in B$ **do**
    **if** $b_{\text{load}} + w_i \leq \max_{\text{load}}$ **then**
      $b^e := b \cup \{i\}$
      $O := O \cup \{b^e\}$
    **end if**
  **end for**
  **if** $O$ is non-empty **then**
    $b^e :=$ ChooseOption$(O, d_{\text{rate}}, l_{\text{size}})$
    $B := B \setminus \{b\} \cup \{b^e\}$
  **else**
    $b^{\text{new}} := \{i\}$ *#Comment: a new bin $b^{\text{new}}$ containing item $i$ is created*
    $B := B \cup \{b^{\text{new}}\}$
  **end if**
**end for**
Sort$(B)$
$S :=$ ExtractSolutionComponents$(B)$
**output:** $S$

#### 3.4.2 Subinstance generation and solving

In the *solve* step, we first generate a reduced problem instance on the basis of $C'$, which is done by adding—for all $i = 1, \ldots, n$—the following constraints to model $\texttt{ILP}^{\text{VSBPP}}_{\text{std}}$ before applying an ILP solver:

1. For all $i, j = 1, \ldots, n$:
   - If $cx^0_{ij} \in C'$ and $cx^1_{ij} \notin C'$: add constraint $x_{ij} = 0$ to $\texttt{ILP}^{\text{VSBPP}}_{\text{std}}$
   - If $cx^0_{ij} \notin C'$ and $cx^1_{ij} \in C'$: add constraint $x_{ij} = 1$ to $\texttt{ILP}^{\text{VSBPP}}_{\text{std}}$
2. For all $j = 1, \ldots, n$ and $k = 1, \ldots, m$:
   - If $cy^0_{jk} \in C'$ and $cy^1_{jk} \notin C'$: add constraint $y_{jk} = 0$ to $\texttt{ILP}^{\text{VSBPP}}_{\text{std}}$
   - If $cy^0_{jk} \notin C'$ and $cy^1_{jk} \in C'$: add constraint $y_{jk} = 1$ to $\texttt{ILP}^{\text{VSBPP}}_{\text{std}}$

In other words, whenever subinstance $C'$ only contains one of the two solution components corresponding to a variable, the value of this variable is fixed to the corresponding value. As a consequence, the more such constraints are added to the original ILP model, the smaller becomes the search space to be explored by the ILP solver for solving the subinstance.

### 3.5 Application of set-covering based CMSA to the VSBPP

The ILP model for solving subinstances in this variant of CMSA—henceforth labeled Adapt-CMSA- SETCOV—is model $\texttt{ILP}^{\text{VSBPP}}_{\text{setcov}}$ from Sect. 3.3. In this case, the complete set of solution components $C$ consists of a component $c^b$ for each valid bin $b$ from $\mathscr{B}$ (see Sect. 3.3), that is, $C := \{c^b \mid b \in \mathscr{B}\}$. Any subset $S \subset C$ such that each item $i \in \{1, \ldots, n\}$ appears in exactly one bin $b$ such that $c^b \in S$ is a valid solution to the tackled VSBPP problem instance.

The probabilistic construction of solutions works exactly in the same way as outlined in Sect. 3.4.1. The only difference lies in the implementation of function ExtractSolutionComponents($B$) that assembles the solution components corresponding to a set of bins $B$. Here, this function simply adds for each $b \in B$ the corresponding solution component $c^b$ to $S$.

The ILP model solved in the *solve step* of this CMSA variant is obtained by exchanging $\mathscr{B}$ in model $\texttt{ILP}^{\text{VSBPP}}_{\text{setcov}}$ by $C'$, that is, by replacing the set of all possible valid bins with the set of those bins that form part of the current subinstance $C'$. The solution $S$ obtained from the ILP solver after $t_{\text{ILP}}$ CPU seconds is then checked for duplicate occurrences of items. If this happens, duplicate items are randomly removed from the bins in $S$ until each item appears exactly once in the bins of $S$.

Finally, extracting the solution components corresponding to the bins in set $B$ in function ExtractSolutionComponents($B$) in line 23 of Algorithm 2 consists simply in adding for each $b \in B$ the corresponding solution component $c^b$ to $S$.

### 3.6 Experimental evaluation

The proposed algorithms were implemented in C++ and the experiments were conducted using a cluster of machines equipped with Intel® Xeon® 5670 CPUs having 12 cores of 2.933 GHz and at least 32 GB of RAM. To solve the respective subinstances within both versions of CMSA, we used CPLEX version 22.1 in one-threaded mode.

#### 3.6.1 Problem instances

The used set of problem instances was introduced in (Haouari & Serairi, 2009). In this set, the number of bin types ($m$) is 7 and bin capacities are defined as $W_1 = 70$, $W_2 = 100$, $W_3 = 130$, $W_4 = 160$, $W_5 = 190$, $W_6 = 220$ and $W_7 = 250$. Moreover, item weights are randomly drawn from $[1, 250]$. In fact, this benchmark set is composed of three different classes of instances. In particular, class B1 is characterized by a linear bin cost function $C_i = W_i$ ($i = 1, \ldots, 7$), class B2 has a concave cost function $C_i = \lceil 10\sqrt{W_i} \rceil$ ($i = 1, \ldots, 7$), and class B3 has a convex cost function $C_i = \lceil 0.1{W_i}^{3/2} \rceil$ ($i = 1, \ldots, 7$). There are 10 problem instances for each combination of $n \in \{100, 200, 500, 1000, 2000\}$ (number of items) and bin cost function class. This makes a total of 150 problem instances. Optimal solutions to these instances are not known.

#### 3.6.2 Parameter tuning

Both CMSA- STD and CMSA- SETCOV require well-working parameter values. In fact, the same five parameters as shown in the first column of Table 1 must be tuned in both cases. The domains that we allowed for these parameters are indicated in the second column of the same table. Parameter tuning was conducted with the `irace` tool (López-Ibáñez et al., 2016). Each algorithm variant was tuned separately for each instance class, with a budget of 2000 runs. Moreover, each run was limited to 150 CPU seconds, as in (Hemmelmayr et al., 2012). The obtained parameter values, as shown in Table 1, were used for the final experimentation.

#### 3.6.3 Numerical results

The results of the current state-of-the-art technique (VNS) from (Hemmelmayr et al., 2012), CMSA- STD, and CMSA- SETCOV are provided in Table 2 (instances of class B1),

**Table 1** Parameter settings for CMSA- STD and CMSA- SETCOV for the three classes of instances

| Parameter | Domain | Class B1 | | Class B2 | | Class B3 | |
|---|---|---|---|---|---|---|---|
| | | CMSA-STD | CMSA-SETCOV | CMSA-STD | CMSA-SETCOV | CMSA-STD | CMSA-SETCOV |
| $t_{\text{ILP}}$ | $(0.3, 30.0)$ | 0.477 | 15.206 | 0.355 | 23.122 | 7.017 | 28.067 |
| $l_{\text{size}}$ | $[1, 10]$ | 1 | 10 | 1 | 5 | 1 | 8 |
| $d_{\text{rate}}$ | $(0.0, 0.99)$ | 0.589 | 0.701 | 0.957 | 0.639 | 0.886 | 0.974 |
| $n_{\text{a}}$ | $[2, 50]$ | 50 | 13 | 46 | 50 | 50 | 46 |
| $age_{\max}$ | $[1, 10]$ | 8 | 3 | 3 | 1 | 3 | 1 |

**Table 2** Results for the 50 instances of class B1 (linear cost function)

| | | best | VNS | | | CMSA-STD | | | CMSA-SETCOV | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| n | # | known | best | avg | avg. time | best | avg | avg. time | best | avg | avg. time |
| 100 | 1 | 12700 | **12700** | 12700.00 | 0.64 | 12760 | 12776.00 | 44.82 | **12700** | 12700.00 | 0.12 |
| 100 | 2 | 12140 | **12140** | 12141.00 | 49.70 | 12210 | 12232.00 | 44.11 | **12140** | 12140.00 | 4.06 |
| 100 | 3 | 13620 | **13620** | 13620.00 | 0.39 | 13670 | 13686.00 | 49.34 | **13620** | 13620.00 | 0.11 |
| 100 | 4 | 12550 | **12550** | 12550.00 | 0.36 | 12620 | 12632.00 | 74.98 | **12550** | 12550.00 | 0.09 |
| 100 | 5 | 10630 | **10630** | 10638.00 | 4.97 | 10690 | 10706.00 | 66.26 | **10630** | 10630.00 | 0.59 |
| 100 | 6 | 11130 | 11140 | 11143.00 | 40.20 | 11190 | 11207.00 | 67.30 | **11130** | 11130.00 | 12.89 |
| 100 | 7 | 13020 | **13020** | 13020.00 | 0.50 | 13080 | 13086.00 | 64.28 | **13020** | 13020.00 | 0.06 |
| 100 | 8 | 12180 | 12180 | 12180.00 | 3.55 | 12260 | 12269.00 | 63.62 | **12170** | 12170.00 | 6.25 |
| 100 | 9 | 11090 | **11090** | 11094.00 | 29.67 | 11180 | 11187.00 | 41.38 | **11090** | 11090.00 | 1.09 |
| 100 | 10 | 12800 | **12800** | 12800.00 | 0.96 | 12870 | 12879.00 | 57.16 | **12800** | 12800.00 | 0.11 |
| 200 | 1 | 25430 | 25440 | 25441.00 | 18.00 | 25640 | 25659.00 | 76.60 | **25430** | 25430.00 | 1.12 |
| 200 | 2 | 26300 | **26300** | 26300.00 | 1.62 | 26400 | 26439.00 | 70.39 | **26300** | 26300.00 | 0.08 |
| 200 | 3 | 27770 | **27770** | 27770.00 | 0.90 | **27770** | 27790.00 | 58.89 | **27770** | 27770.00 | 0.03 |
| 200 | 4 | 24300 | 24300 | 24302.00 | 46.50 | 24490 | 24507.00 | 70.86 | **24290** | 24290.00 | 5.88 |
| 200 | 5 | 25820 | **25820** | 25820.00 | 0.87 | 25940 | 25968.00 | 59.44 | **25820** | 25820.00 | 0.03 |
| 200 | 6 | 23820 | 23820 | 23829.00 | 39.22 | 23980 | 24007.00 | 60.76 | **23810** | 23810.00 | 1.05 |
| 200 | 7 | 28590 | **28590** | 28590.00 | 0.83 | 28600 | 28624.00 | 55.83 | **28590** | 28590.00 | 0.03 |
| 200 | 8 | 25900 | **25900** | 25900.00 | 0.74 | 26040 | 26067.00 | 83.77 | **25900** | 25900.00 | 0.21 |
| 200 | 9 | 24890 | **24890** | 24899.00 | 23.85 | 25070 | 25098.00 | 48.45 | **24890** | 24890.00 | 0.22 |
| 200 | 10 | 25760 | **25760** | 25760.00 | 1.21 | 25850 | 25868.00 | 79.21 | **25760** | 25760.00 | 0.05 |
| 500 | 1 | 61770 | 61810 | 61826.00 | 48.74 | 62350 | 62424.00 | 95.47 | **61750** | 61758.00 | 20.56 |
| 500 | 2 | 62090 | 62090 | 62103.00 | 71.15 | 62560 | 62628.00 | 59.03 | **62070** | 62070.00 | 11.65 |
| 500 | 3 | 66770 | 66770 | 66793.00 | 76.74 | 67320 | 67371.00 | 60.61 | **66760** | 66763.00 | 31.65 |
| 500 | 4 | 63970 | **63970** | 63987.00 | 30.78 | 64360 | 64410.00 | 75.02 | **63970** | 63970.00 | 2.32 |
| 500 | 5 | 62150 | 62180 | 62182.00 | 24.33 | 62670 | 62698.00 | 80.46 | **62150** | 62150.00 | 0.44 |
| 500 | 6 | 61130 | 61130 | 61142.00 | 25.35 | 61670 | 61702.00 | 51.70 | **61090** | 61090.00 | 24.31 |
| 500 | 7 | 63340 | 63340 | 63351.00 | 16.22 | 63930 | 63991.00 | 64.20 | **63320** | 63320.00 | 2.14 |
| 500 | 8 | 63250 | 63250 | 63269.00 | 77.18 | 63760 | 63809.00 | 97.81 | **63210** | 63210.00 | 4.72 |
| 500 | 9 | 61170 | 61170 | 61172.00 | 35.38 | 61740 | 61777.00 | 78.08 | **61120** | 61120.00 | 36.41 |
| 500 | 10 | 62000 | 62000 | 62011.00 | 27.53 | 62540 | 62562.00 | 69.21 | **61990** | 61990.00 | 12.14 |
| 1000 | 1 | 126610 | 126610 | 126649.00 | 72.29 | 127620 | 127674.00 | 73.63 | **126490** | 126496.00 | 52.58 |
| 1000 | 2 | 123250 | 123290 | 123332.00 | 28.33 | 124370 | 124466.00 | 69.56 | **123120** | 123120.00 | 13.62 |
| 1000 | 3 | 123070 | 123070 | 123093.00 | 29.80 | 124320 | 124390.00 | 54.98 | **123020** | 123029.00 | 36.97 |
| 1000 | 4 | 127370 | 127370 | 127370.00 | 5.14 | 128510 | 128570.00 | 66.89 | **127360** | 127360.00 | 24.74 |
| 1000 | 5 | 127710 | 127710 | 127710.00 | 4.74 | 128990 | 129036.00 | 74.28 | **127660** | 127660.00 | 23.94 |
| 1000 | 6 | 125580 | 125580 | 125617.00 | 56.97 | 126640 | 126766.00 | 52.26 | **125520** | 125522.00 | 56.93 |
| 1000 | 7 | 128260 | **128260** | 128260.00 | 27.97 | 129290 | 129380.00 | 62.90 | **128260** | 128260.00 | 1.49 |
| 1000 | 8 | 130410 | **130410** | 130422.00 | 22.44 | 131450 | 131513.00 | 65.99 | **130410** | 130410.00 | 5.11 |
| 1000 | 9 | 125680 | 125680 | 125692.00 | 32.00 | 126910 | 126970.00 | 68.94 | **125630** | 125635.00 | 28.76 |
| 1000 | 10 | 129400 | 129400 | 129400.00 | 6.41 | 130380 | 130480.00 | 62.26 | **129380** | 129380.00 | 7.77 |
| 2000 | 1 | 254330 | 254330 | 254370.00 | 39.03 | 256380 | 256455.00 | 59.79 | **254290** | 254290.00 | 44.52 |
| 2000 | 2 | 257370 | 257390 | 257390.00 | 8.21 | 259590 | 259723.00 | 73.83 | **257330** | 257330.00 | 47.29 |
| 2000 | 3 | 251880 | 251880 | 251883.00 | 29.11 | 254100 | 254180.00 | 80.85 | **251770** | 251770.00 | 57.39 |
| 2000 | 4 | 248520 | 248520 | 248547.00 | 115.29 | 250610 | 250653.00 | 68.64 | **248470** | 248470.00 | 26.36 |
| 2000 | 5 | 245110 | 245110 | 245137.00 | 40.76 | 247810 | 247905.00 | 46.48 | **245060** | 245066.00 | 88.49 |
| 2000 | 6 | 250930 | 250930 | 250951.00 | 29.41 | 253500 | 253571.00 | 69.33 | **250870** | 250878.00 | 40.58 |
| 2000 | 7 | 258700 | 258700 | 258725.00 | 79.46 | 261140 | 261198.00 | 53.64 | **258680** | 258690.00 | 62.45 |
| 2000 | 8 | 256950 | **256950** | 256950.00 | 9.18 | 259330 | 259412.00 | 49.21 | 256970 | 256978.00 | 71.54 |
| 2000 | 9 | 258480 | 258480 | 258480.00 | 6.43 | 260720 | 260866.00 | 56.24 | **258450** | 258458.00 | 66.39 |
| 2000 | 10 | 255750 | **255750** | 255751.00 | 10.72 | 257670 | 257802.00 | 71.50 | **255750** | 255750.00 | 6.92 |

**Table 3** Results for the 50 instances of class B2 (concave cost function)

| | | best | VNS | | | CMSA-STD | | | CMSA-SETCOV | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| n | # | known | best | avg | avg. time | best | avg | avg. time | best | avg | avg. time |
| 100 | 1 | 8890 | **8890** | 8890.00 | 0.02 | 8920 | 8923.90 | 63.88 | **8890** | 8890.00 | 0.04 |
| 100 | 2 | 7832 | **7832** | 7832.00 | 8.97 | 7864 | 7870.90 | 72.02 | **7832** | 7832.00 | 2.53 |
| 100 | 3 | 8516 | **8516** | 8516.00 | 2.18 | **8516** | 8545.10 | 97.94 | **8516** | 8516.00 | 0.04 |
| 100 | 4 | 8591 | **8591** | 8591.00 | 0.28 | 8611 | 8619.50 | 78.30 | **8591** | 8591.00 | 0.02 |
| 100 | 5 | 8474 | **8474** | 8474.00 | 0.00 | 8496 | 8502.00 | 68.44 | **8474** | 8474.00 | 0.28 |
| 100 | 6 | 7538 | **7538** | 7538.00 | 40.28 | 7571 | 7578.00 | 53.49 | **7538** | 7538.00 | 3.59 |
| 100 | 7 | 7876 | **7876** | 7876.00 | 0.00 | 7890 | 7898.00 | 47.32 | **7876** | 7876.00 | 0.03 |
| 100 | 8 | 8116 | **8116** | 8119.20 | 28.20 | 8148 | 8158.90 | 58.86 | **8116** | 8116.00 | 0.70 |
| 100 | 9 | 8392 | **8392** | 8392.00 | 0.14 | 8409 | 8414.90 | 77.77 | **8392** | 8392.00 | 0.43 |
| 100 | 10 | 9127 | **9127** | 9127.00 | 0.00 | 9137 | 9139.00 | 53.29 | **9127** | 9127.00 | 0.03 |
| 200 | 1 | 17307 | **17307** | 17307.00 | 4.50 | 17445 | 17453.10 | 80.09 | **17307** | 17307.00 | 4.53 |
| 200 | 2 | 16391 | 16392 | 16396.50 | 50.71 | 16533 | 16562.10 | 84.47 | **16391** | 16394.30 | 62.26 |
| 200 | 3 | 16637 | 16637 | 16637.00 | 0.11 | 16687 | 16708.70 | 63.75 | **16629** | 16629.80 | 38.82 |
| 200 | 4 | 15864 | **15864** | 15864.00 | 13.95 | 15925 | 15953.50 | 77.29 | **15864** | 15864.00 | 0.30 |
| 200 | 5 | 17699 | **17699** | 17699.00 | 0.05 | 17729 | 17735.50 | 84.51 | **17699** | 17699.00 | 0.04 |
| 200 | 6 | 15457 | **15457** | 15459.10 | 50.19 | 15541 | 15572.40 | 73.06 | **15457** | 15457.00 | 3.03 |
| 200 | 7 | 16203 | **16203** | 16203.00 | 0.04 | 16329 | 16343.80 | 85.28 | **16203** | 16203.00 | 0.08 |
| 200 | 8 | 15353 | **15353** | 15371.70 | 26.90 | 15490 | 15503.70 | 82.03 | **15353** | 15353.00 | 2.49 |
| 200 | 9 | 15860 | **15860** | 15860.00 | 28.71 | 16062 | 16072.40 | 78.29 | **15860** | 15860.00 | 0.11 |
| 200 | 10 | 15292 | **15292** | 15292.00 | 2.03 | 15437 | 15447.90 | 81.29 | **15292** | 15292.00 | 0.14 |
| 500 | 1 | 39307 | 39307 | 39310.10 | 51.37 | 39555 | 39609.40 | 81.73 | **39305** | 39305.60 | 46.36 |
| 500 | 2 | 40767 | 40767 | 40774.90 | 1.61 | 41027 | 41085.90 | 74.65 | **40765** | 40765.00 | 9.64 |
| 500 | 3 | 39963 | 39964 | 39964.00 | 1.79 | 40273 | 40312.70 | 79.02 | **39963** | 39963.00 | 2.19 |
| 500 | 4 | 38945 | 38992 | 39001.60 | 12.81 | 39391 | 39439.80 | 76.98 | **38934** | 38934.70 | 32.24 |
| 500 | 5 | 39785 | 39785 | 39788.30 | 4.47 | 40122 | 40166.60 | 81.12 | **39775** | 39775.00 | 1.70 |
| 500 | 6 | 43096 | **43096** | 43096.00 | 0.32 | 43213 | 43288.10 | 99.02 | **43096** | 43096.00 | 10.50 |
| 500 | 7 | 41307 | 41307 | 41319.90 | 51.57 | 41541 | 41574.10 | 78.85 | **41306** | 41306.00 | 7.00 |
| 500 | 8 | 39756 | 39756 | 39758.00 | 4.73 | 40049 | 40080.40 | 70.89 | **39738** | 39741.40 | 49.30 |
| 500 | 9 | 40166 | 40191 | 40200.10 | 23.69 | 40424 | 40480.00 | 73.82 | **40154** | 40154.00 | 26.21 |
| 500 | 10 | 41046 | **41046** | 41046.00 | 19.67 | 41324 | 41352.80 | 63.15 | **41046** | 41046.00 | 0.28 |
| 1000 | 1 | 81458 | 81458 | 81458.60 | 0.01 | 81899 | 81988.20 | 90.45 | **81447** | 81447.00 | 0.82 |
| 1000 | 2 | 78523 | 78593 | 78593.80 | 16.24 | 79309 | 79380.20 | 65.58 | **78515** | 78518.50 | 100.57 |
| 1000 | 3 | 81544 | 81619 | 81623.80 | 28.25 | 82216 | 82319.70 | 65.93 | **81525** | 81528.50 | 101.22 |
| 1000 | 4 | 80265 | 80265 | 80278.30 | 0.02 | 80813 | 80920.60 | 65.70 | **80255** | 80259.40 | 48.64 |
| 1000 | 5 | 81076 | 81132 | 81177.20 | 21.19 | 81795 | 81844.50 | 83.11 | **81066** | 81070.00 | 55.43 |
| 1000 | 6 | 81333 | **81333** | 81335.00 | 35.23 | 81879 | 82000.90 | 89.12 | 81343 | 81348.00 | 44.53 |
| 1000 | 7 | 81200 | 81200 | 81200.00 | 34.60 | 81555 | 81715.00 | 90.93 | **81199** | 81199.00 | 2.76 |
| 1000 | 8 | 80899 | 81001 | 81002.20 | 53.20 | 81658 | 81747.20 | 106.88 | **80849** | 80856.20 | 28.34 |
| 1000 | 9 | 78381 | 78381 | 78384.30 | 25.60 | 79037 | 79160.40 | 67.42 | **78365** | 78374.50 | 106.69 |
| 1000 | 10 | 84535 | 84535 | 84550.60 | 71.40 | 84853 | 84931.20 | 56.74 | **84503** | 84504.80 | 71.25 |
| 2000 | 1 | 160446 | 160488 | 160489.70 | 27.49 | 161725 | 161903.40 | 92.55 | **160287** | 160290.50 | 107.32 |
| 2000 | 2 | 162193 | **162193** | 162193.00 | 3.89 | 163211 | 163370.10 | 62.43 | **162193** | 162197.00 | 46.56 |
| 2000 | 3 | 161879 | 161935 | 161935.00 | 0.08 | 163086 | 163230.00 | 91.45 | **161857** | 161861.30 | 93.70 |
| 2000 | 4 | 161128 | 161128 | 161132.00 | 41.93 | 161945 | 162080.00 | 62.24 | **161064** | 161064.70 | 46.03 |
| 2000 | 5 | 164625 | 164625 | 164628.10 | 24.28 | 165530 | 165647.80 | 74.98 | **164605** | 164612.90 | 95.17 |
| 2000 | 6 | 159107 | 159107 | 159110.60 | 45.04 | 160338 | 160419.90 | 70.71 | **159096** | 159096.00 | 40.79 |
| 2000 | 7 | 162445 | 162448 | 162448.20 | 36.93 | 163422 | 163544.70 | 88.91 | **162391** | 162391.00 | 12.01 |
| 2000 | 8 | 159878 | 159878 | 159878.00 | 1.79 | 161171 | 161238.50 | 81.00 | **159869** | 159878.20 | 47.41 |
| 2000 | 9 | 161694 | 161712 | 161712.00 | 0.00 | 162638 | 162748.00 | 72.75 | **161683** | 161683.00 | 45.61 |
| 2000 | 10 | 153403 | 153571 | 153576.70 | 64.29 | 154768 | 154908.70 | 71.44 | **153266** | 153267.00 | 57.66 |

Table 3 (instances of class B2), and Table 4 (instances of class B3). The first two columns of these tables indicate the number of items ($n$) and the instance number (#), respectively. For each problem instance (table row), the results of the three algorithms are provided in terms of the best solution found over 10 runs, the average of the best solutions from the 10 runs, and the average times at which these solutions were found within the time limit of 150 CPU seconds per run. The following observations can be made:

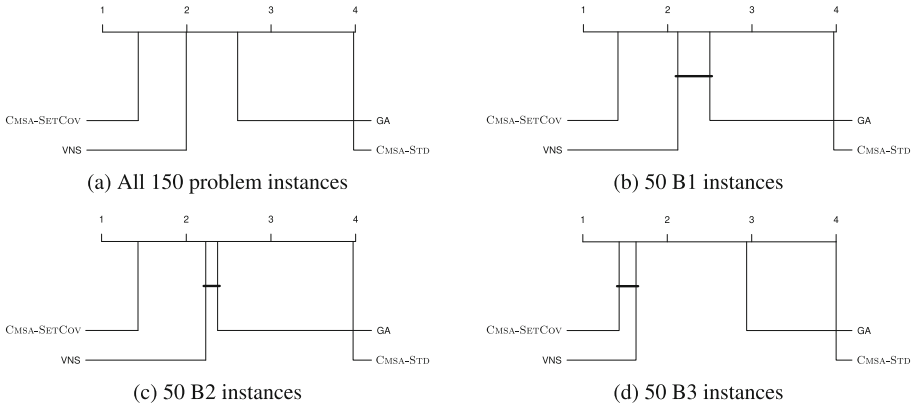
- First, CMSA- SETCOV clearly outperforms both CMSA- STD and VNS. Only in two out of 150 cases, CMSA- STD is able to find a solution of the same quality as the one found by CMSA- SETCOV. Moreover, while CMSA- SETCOV and VNS perform comparably for rather small problem instances with $n \in \{100, 200\}$ items, CMSA- SETCOV clearly outperforms VNS in the context of larger problem instances (especially for $n \in \{1000, 2000\}$).

**Table 4** Results for the 50 instances of class B3 (convex cost function)

| | | best | VNS | | | CMSA-STD | | | CMSA-SETCOV | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| n | # | known | best | avg | avg. time | best | avg | avg. time | best | avg | avg. time |
| 100 | 1 | 19364 | **19364** | 19364.00 | 0.30 | 19393 | 19414.80 | 79.77 | **19364** | 19364.00 | 0.02 |
| 100 | 2 | 19000 | **19000** | 19000.00 | 0.20 | 19016 | 19025.10 | 32.03 | **19000** | 19000.00 | 0.03 |
| 100 | 3 | 18272 | **18272** | 18272.00 | 0.10 | 18280 | 18298.50 | 41.17 | **18272** | 18272.00 | 0.01 |
| 100 | 4 | 19016 | **19016** | 19016.00 | 0.50 | 19029 | 19054.30 | 48.91 | **19016** | 19016.00 | 0.02 |
| 100 | 5 | 16612 | **16612** | 16612.00 | 0.15 | 16648 | 16653.20 | 71.84 | **16612** | 16612.00 | 0.04 |
| 100 | 6 | 18632 | **18632** | 18632.00 | 0.02 | 18649 | 18662.90 | 60.51 | **18632** | 18632.00 | 0.06 |
| 100 | 7 | 18682 | **18682** | 18682.00 | 0.26 | 18708 | 18726.00 | 92.06 | **18682** | 18682.00 | 0.04 |
| 100 | 8 | 19517 | **19517** | 19517.00 | 0.14 | 19520 | 19540.00 | 78.12 | **19517** | 19517.00 | 0.02 |
| 100 | 9 | 17950 | **17950** | 17950.00 | 0.21 | 17968 | 17983.90 | 92.37 | **17950** | 17950.00 | 0.03 |
| 100 | 10 | 17127 | **17127** | 17127.00 | 0.50 | 17135 | 17150.30 | 59.36 | **17127** | 17127.00 | 0.02 |
| 200 | 1 | 35423 | **35423** | 35423.40 | 32.29 | 35600 | 35636.60 | 56.79 | **35423** | 35423.00 | 0.79 |
| 200 | 2 | 36362 | **36362** | 36362.00 | 54.19 | 36638 | 36699.60 | 70.19 | **36362** | 36362.00 | 6.80 |
| 200 | 3 | 33390 | **33390** | 33390.00 | 1.96 | 33620 | 33648.20 | 81.67 | **33390** | 33390.00 | 0.30 |
| 200 | 4 | 34327 | **34327** | 34327.00 | 2.14 | 34529 | 34572.10 | 76.59 | **34327** | 34327.00 | 0.13 |
| 200 | 5 | 38055 | **38055** | 38055.00 | 4.96 | 38231 | 38265.50 | 76.52 | **38055** | 38055.00 | 0.09 |
| 200 | 6 | 35009 | **35009** | 35009.00 | 2.96 | 35194 | 35228.90 | 42.86 | **35009** | 35009.00 | 0.10 |
| 200 | 7 | 38175 | **38175** | 38175.00 | 3.94 | 38313 | 38353.40 | 69.01 | **38175** | 38175.00 | 0.47 |
| 200 | 8 | 36003 | **36003** | 36003.00 | 2.84 | 36154 | 36180.40 | 80.27 | **36003** | 36003.00 | 0.09 |
| 200 | 9 | 32700 | **32700** | 32700.80 | 55.99 | 32883 | 32922.50 | 61.19 | **32700** | 32700.00 | 1.75 |
| 200 | 10 | 36998 | **36998** | 36998.00 | 0.28 | 37124 | 37236.70 | 57.45 | **36998** | 36998.00 | 0.18 |
| 500 | 1 | 94768 | **94768** | 94768.80 | 73.74 | 95293 | 95363.90 | 114.88 | **94768** | 94768.00 | 1.11 |
| 500 | 2 | 97983 | **97983** | 97983.00 | 79.44 | 98455 | 98506.40 | 71.85 | **97983** | 97983.00 | 0.84 |
| 500 | 3 | 95832 | **95832** | 95832.60 | 63.45 | 96365 | 96537.10 | 90.70 | **95832** | 95832.00 | 1.22 |
| 500 | 4 | 91068 | **91068** | 91068.00 | 35.51 | 91598 | 91723.30 | 96.05 | **91068** | 91068.00 | 0.45 |
| 500 | 5 | 87676 | **87676** | 87676.00 | 58.47 | 88479 | 88509.70 | 85.86 | **87676** | 87676.00 | 0.63 |
| 500 | 6 | 83124 | **83124** | 83124.00 | 32.00 | 83926 | 84065.90 | 63.61 | **83124** | 83124.00 | 21.24 |
| 500 | 7 | 90407 | **90407** | 90408.00 | 93.58 | 91061 | 91124.80 | 32.83 | **90407** | 90407.00 | 0.97 |
| 500 | 8 | 87059 | **87059** | 87059.00 | 23.34 | 87844 | 87883.40 | 57.71 | **87059** | 87059.00 | 7.37 |
| 500 | 9 | 87398 | **87398** | 87398.00 | 32.00 | 88012 | 88148.40 | 105.40 | **87398** | 87398.00 | 2.50 |
| 500 | 10 | 90541 | **90541** | 90543.20 | 72.71 | 91097 | 91226.80 | 86.41 | 90543 | 90543.00 | 23.88 |
| 1000 | 1 | 176950 | **176950** | 176954.00 | 122.51 | 178524 | 178603.40 | 64.64 | 176953 | 176955.30 | 98.43 |
| 1000 | 2 | 180993 | 180993 | 180996.20 | 100.89 | 182387 | 182477.00 | 73.19 | **180989** | 180990.60 | 73.80 |
| 1000 | 3 | 182758 | 182758 | 182760.60 | 128.28 | 184448 | 184540.30 | 52.71 | **182754** | 182754.60 | 58.44 |
| 1000 | 4 | 180859 | 180859 | 180863.80 | 99.56 | 182523 | 182593.60 | 71.58 | **180857** | 180859.30 | 58.48 |
| 1000 | 5 | 179158 | 179158 | 179160.80 | 114.74 | 180722 | 180766.40 | 98.23 | **179154** | 179154.80 | 61.79 |
| 1000 | 6 | 188838 | **188838** | 188841.60 | 109.85 | 190405 | 190462.00 | 74.14 | 188839 | 188840.20 | 34.81 |
| 1000 | 7 | 178185 | 178185 | 178189.30 | 104.86 | 179873 | 179929.90 | 69.07 | **178183** | 178189.70 | 89.80 |
| 1000 | 8 | 177461 | 177461 | 177466.60 | 121.33 | 179120 | 179188.00 | 88.13 | **177459** | 177459.00 | 14.70 |
| 1000 | 9 | 181005 | 181005 | 181010.80 | 111.06 | 182688 | 182735.90 | 67.75 | **181001** | 181007.70 | 69.93 |
| 1000 | 10 | 176902 | **176902** | 176904.40 | 97.87 | 178608 | 178668.80 | 90.00 | **176902** | 176917.20 | 49.65 |
| 2000 | 1 | 356244 | 356244 | 356249.30 | 129.75 | 360181 | 360299.10 | 68.42 | **356235** | 356236.60 | 85.25 |
| 2000 | 2 | 369839 | 369839 | 369852.90 | 142.05 | 373372 | 373580.80 | 57.24 | **369811** | 369831.70 | 68.47 |
| 2000 | 3 | 364550 | 364550 | 364559.70 | 131.84 | 368194 | 368358.80 | 80.45 | **364527** | 364562.30 | 72.10 |
| 2000 | 4 | 356984 | 356984 | 356995.20 | 131.33 | 360844 | 360947.80 | 88.17 | **356958** | 356969.80 | 102.76 |
| 2000 | 5 | 365557 | **365557** | 365572.70 | 127.77 | 369038 | 369166.60 | 79.64 | 365564 | 365576.40 | 106.84 |
| 2000 | 6 | 365142 | 365142 | 365159.60 | 133.58 | 368915 | 369041.40 | 78.61 | **365116** | 365120.80 | 56.37 |
| 2000 | 7 | 360824 | 360824 | 360826.80 | 138.66 | 364687 | 364771.30 | 67.67 | **360816** | 360838.20 | 90.21 |
| 2000 | 8 | 371799 | 371799 | 371804.50 | 136.52 | 375086 | 375321.60 | 64.33 | **371779** | 371798.60 | 96.17 |
| 2000 | 9 | 355723 | **355723** | 355734.10 | 137.09 | 359485 | 359606.10 | 73.22 | 355726 | 355745.30 | 104.49 |
| 2000 | 10 | 357058 | 357058 | 357069.90 | 131.26 | 361201 | 361305.50 | 75.07 | **357036** | 357040.10 | 68.10 |

- CMSA- SETCOV is able to find new best-known solutions in 68 out of 150 cases. In particular, CMSA- SETCOV finds 27 new best-known solutions in the case of the 50 B1 instances, 26 new best-known solutions in the case of the 50 B2 instances, and 15 new best-known solutions in the case of the 50 B3 instances.
- Only in 7 out of 150 cases, the best solution found by CMSA- SETCOV is slightly worse than the one found by VNS.

In order to test the statistical relevance of these results we produced so-called critical difference (CD) plots with the `scmamp` toolkit Calvo and Santafe (2016) of the R statistical language, which implements the procedure recommended in García and Herrera (2008) for the comparison of multiple algorithms over multiple problem instances. The horizontal axis of such a CD plot shows the range of algorithm ranks, while each of the vertical




(a) All 150 problem instances (b) 50 B1 instances

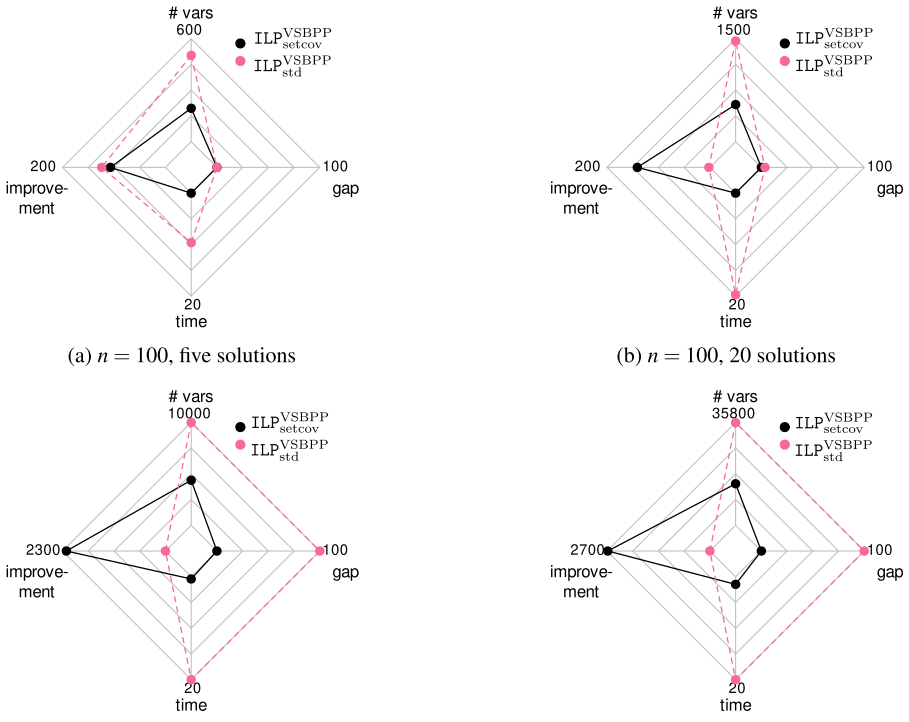
(c) 50 B2 instances (d) 50 B3 instances

**Fig. 3** Criticial difference (CD) plots showing the statistical significance of the results

lines represents the average rank of the corresponding algorithm for the considered instance group. Bold horizontal lines connecting algorithm markers indicate that the corresponding algorithms perform statistically equivalent–i.e. the critical difference is not greater than the significance level of 0.05. Note also that, in the context of these plots, we added the results of the GA approach from (Crainic et al., 2011). The plot from Fig. 3a shows that–from a global point of view–CMSA- SETCOV outperforms all other algorithms with statistical significance. When considering the instances from the three classes separately, no statistically significant difference between CMSA- SETCOV and VNS can be detected in the context of the B3 class (see Fig. 3d).

### 3.6.4 Performance Difference Between the two VSBPP ILP Models

Finally, we aim to show why CMSA- SETCOV outperforms CMSA- STD so clearly. For this purpose, we generate subinstances of different sizes, translate them both into models $\texttt{ILP}_{\text{std}}^{\text{VSBPP}}$ and $\texttt{ILP}_{\text{setcov}}^{\text{VSBPP}}$, and solve them with CPLEX. In particular, we generated 10 subinstances by probabilistically constructing five, respectively, 20 solutions and by merging their solution components in order to obtain subinstances. This was done for one B1 instance with $n = 100$ items and for another B1 instance with $n = 2000$ items. Figure 4 shows radar charts that present the obtained results in the four different cases. Each radar chart provides four different measures, averaged over 10 runs: (1) the number of variables in the models of the subinstances (top), (2) the relative MIP gap after termination of CPLEX (right), (3) the computation time required by CPLEX (bottom), and (4) the absolute improvement when comparing the result of solving the subinstance with the best individual solution that was used to generate the subinstance. Note that the time limit for CPLEX was set to 20 CPU seconds in all cases. In this context, a model is promising if the improvement (left) is large, and the number of variables (top), the relative MIP gap (right) and the required time (bottom) are low. The four radar plots indicate that this is indeed the case for model $\texttt{ILP}_{\text{setcov}}^{\text{VSBPP}}$, while the opposite is actually the case for model $\texttt{ILP}_{\text{std}}^{\text{VSBPP}}$. Obviously, the results become more pronounced with growing problem instance size.




(a) $n = 100$, five solutions

(b) $n = 100$, 20 solutions

(c) $n = 2000$, five solutions

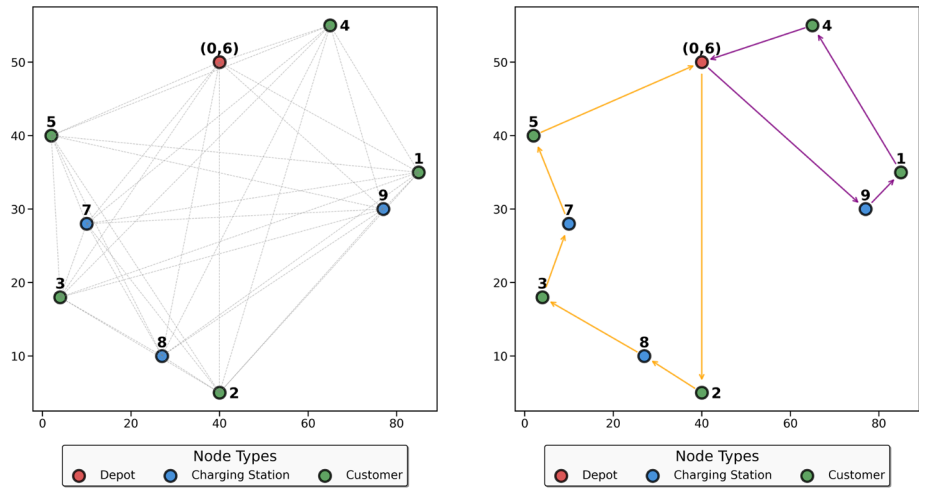
(d) $n = 2000$, 20 solutions

**Fig. 4** Radar charts concerning the comparison of the two ILP models applied to a B1 instance with 100 items (see (**a**) and (**b**)), and to a problem instance with 2000 items (see (**c**) and (**d**))

## 4 Application to an electric vehicle routing problem

As mentioned before, another large family of problems where standard ILP models can be replaced by set-covering-based ILP models is vehicle routing. In recent years, due to environmental concerns, many researchers have focused on vehicle routing problems (VRPs) concerned with electric vehicles, so-called electric vehicle routing problems (EVRPs). In this section, with the aim of presenting a second example for the benefits of using set-covering-based ILP models within CMSA algorithms, we consider the so-called electric vehicle routing problem with time windows and simultaneous pickups and deliveries (EVRP-TW-SPD). In a preliminar work (Akbay et al., 2023), we already presented the application of a CMSA variant called Adapt-CMSA based on the standard assignment-type ILP model of the problem. In this section, we present an improved algorithm version together with the variant that makes use of the set-covering-based ILP model.

The standard ILP model of the EVRP-TW-SPD is an extension of the model for the EVRP-TW-PR problem proposed in (Keskin & Çatay, 2016), which—in turn—is a modified variant of the model for the EVRP-TW problem proposed in (Schneider et al., 2014). In particular, we extend the model further in order to consider SPD constraints. When dealing with SPD constraints in the context of vehicle routing problems, it is important to note that each customer's demand may consist of two distinct requirements: (1) delivering goods to the demand point, known as the *"delivery demand"*, and (2) collecting goods from the demand




(a) EVRP Instance (b) Solution

**Fig. 5** Illustration of an EVRP instance and its solution. **a** presents a map showing the locations of a depot, five customers, and three charging stations based on Cartesian coordinates. Gray dashed lines indicate a fully connected graph connecting each pair of nodes. **b** shows a valid solution to the given instance on the same map, with two distinct tours represented by arrows with different colors. Both routes begin and end at the depot, passing through various customers and charging stations

point, known as the *"pickup demand"*. It is necessary to satisfy both demands simultaneously when a vehicle visits a particular customer.

We adopt the same notation as in the previous works to maintain consistency with the existing literature. In particular, the EVRP-TW-SPD problem involves a set of $N$ customers, denoted by $V = \{1, \ldots, N\}$, and a set of charging stations, denoted by $F$.

As we will model the EVRP-TW-SPD by means of a two-index ILP model, which allows the visit of each node (customers and charging stations) exactly once, we enable multiple visits (possibly by different vehicles) to each charging station by defining a set $F'$ that contains multiple copies of each charging station from $F$. Note that the utilization of such dummy nodes in the context of two-index ILP models is already known from the related literature (Bard et al., 1998; Erdoğan & Miller-Hooks, 2012). However, determining the appropriate number of copies for each charging station is non-trivial. Insufficient replication, which leads to restricting the number of permissible visits to the same charging station, may exclude an optimal solution from the search space. Conversely, an excessive number of replicas could inflate the model size, leading to prolonged execution times for ILP solvers. The way in which the utilized number of charging station copies was determined for each problem instance will be described in Sect. 4.6.3.

The depot is represented by nodes 0 and $N + 1$, where node 0 is the starting point and node $N + 1$ is the ending point for each route. Note that both 0 and $N + 1$ refer to the same depot. The set $V' = V \cup F'$ contains all customers and dummy charging stations. Moreover, sub-indexes 0, $N + 1$, or both, added to a node-set, indicate the inclusion of the respective depot instances to that set. For example, $V_0$ is the node set that contains all customers and the instance 0 of the depot. Based on the above notations, we define the following sets:

1. $F'_0 := F' \cup \{0\}$
2. $V'_0 := V' \cup \{0\}$

3. $V_{N+1}^{'} := V^{'} \cup \{N+1\}$
4. $V_{0,N+1}^{'} := V^{'} \cup \{0\} \cup \{N+1\}$

Following established sets and notations, the EVRP-TW-SPD can be defined on a complete, directed graph $G(V_{0,N+1}^{'}, A)$. $A = \{(i,j)|i,j \in V_{0,N+1}^{'}, i \neq j\}$ is the set of arcs where each arc has a corresponding distance $d_{ij}$ and travel time $t_{ij}$. Note that both distances and travel times are symmetric, that is, it always holds that $d_{ij} = d_{ji}$ and $t_{ij} = t_{ji}$. The energy consumed per unit distance traveled by an electric vehicle (EV) is denoted by a constant $h$. A fleet of electric vehicles with identical loading capacity $EV_{cap}$ and battery capacity $B_{cap}$ is stationed at a depot to satisfy delivery demand $q_i > 0$ and pickup demand $p_i > 0$ of each customer $i$ simultaneously. Each vertex $i \in V_{0,N+1}^{'}$ is only allowed to be visited within a time window $[e_i, l_i]$ that indicates the earliest and latest possible visiting times. Moreover, each customer $i \in V$ has a service time $s_i$, which refers to the time an electric vehicle spends visiting a customer. When an EV visits a charging station, its battery is charged as determined by a parameter $g > 0$, which indicates the units of time required to charge one unit of energy. Note that this way of fixing the charging speed is very usual in EVRPs; see, for example, (Schneider et al., 2014; Keskin & Çatay, 2016).

The following decision variables are used to formulate the problem's ILP model. The binary decision variable $x_{ij}$ takes a value of 1 if the arc $a_{ij}$ is included in the route and 0 otherwise. The starting time of the service for each customer visited by the electric vehicle is monitored by the decision variable $\tau_i$. Moreover, to keep track of the battery's state of charge upon arrival and departure at each vertex $i \in V_{0,N+1}^{'}$, the decision variables $y_i$ and $Y_i$ are employed, respectively. Furthermore, the remaining cargo to be delivered to the customers of the route and the amount of cargo already collected (picked up) at the previously visited customers are represented by the variables $u_{ij}$ and $v_{ij}$, respectively. ILP model $\texttt{ILP}_{\text{std}}^{\text{EVRP}}$ is technically described as follows.

$$\textbf{Min} \quad \sum_{i \in V_0^{'}, j \in V_{N+1}^{'}} d_{ij} x_{ij} + \sum_{j \in V_{N+1}^{'}} M x_{0j} \tag{10}$$

$$\textbf{s.t.} \quad \sum_{j \in V_{N+1}^{'}, i \neq j} x_{ij} = 1 \qquad \forall i \in V \tag{11}$$

$$\sum_{j \in V_{N+1}^{'}, i \neq j} x_{ij} \leq 1 \qquad \forall i \in F^{'} \tag{12}$$

$$\sum_{i \in V_0^{'}, i \neq j} x_{ij} - \sum_{i \in V_{N+1}^{'}, i \neq j} x_{ji} = 0 \qquad \forall j \in V^{'} \tag{13}$$

$$\tau_i + (t_{ij} + s_i) x_{ij} - l_0(1 - x_{ij}) \leq \tau_j \qquad \forall i \in V_0, j \in V_{N+1}^{'}, i \neq j \tag{14}$$

$$\tau_i + t_{ij} x_{ij} + g(Y_i - y_i) - (l_0 + g B_{cap})(1 - x_{ij}) \leq \tau_j \qquad \forall i \in F^{'}, \forall j \in V_{N+1}^{'}, i \neq j \tag{15}$$

$$e_j \leq \tau_j \leq l_j \qquad \forall j \in V_{0,N+1}^{'} \tag{16}$$

$$0 \leq u_{0j} \leq EV_{cap} \qquad \forall j \in V_{N+1}^{'} \tag{17}$$

$$v_{0j} = 0 \qquad \forall j \in V_{N+1}^{'} \tag{18}$$

$$\sum_{i \in V_0^{'}, i \neq j} u_{ij} - \sum_{i \in V_{N+1}^{'}, i \neq j} u_{ji} = q_j \qquad \forall j \in V^{'} \tag{19}$$

$$\sum_{i \in V_{N+1}^{'}, i \neq j} v_{ji} - \sum_{i \in V_0^{'}, i \neq j} v_{ij} = p_j \qquad \forall j \in V^{'} \tag{20}$$

$$u_{ij} + v_{ij} \leq EV_{cap} x_{ij} \qquad \forall i \in V_0^{'}, j \in V_{N+1}^{'}, i \neq j \quad (21)$$

$$0 \leq y_j \leq y_i - (hd_{ij}) x_{ij} + B_{cap}(1 - x_{ij}) \qquad \forall i \in V, \forall j \in V_{N+1}^{'}, i \neq j \quad (22)$$

$$0 \leq y_j \leq Y_i - (hd_{ij}) x_{ij} + B_{cap}(1 - x_{ij}) \qquad \forall i \in F_0^{'}, \forall j \in V_{N+1}^{'}, i \neq j \quad (23)$$

$$y_i \leq Y_i \leq B_{cap} \qquad \forall i \in F_0^{'} \quad (24)$$

$$x_{ij} \in 0, 1 \qquad \forall i \in V_0^{'}, j \in V_{N+1}^{'}, i \neq j \quad (25)$$

The distance-based objective function from (Keskin & Çatay, 2016) is extended in order to prioritize solutions that utilize fewer vehicles, even if the total distance traveled in such cases is greater than in other solutions. This is done by introducing an additional cost parameter $M > 0$ per vehicle utilized. Note that the number of vehicles used in a solution corresponds to the variables on outgoing arcs of the depot (0) that have a value of 1. In this line, the objective function (10) minimizes the total travel and vehicle cost. Constraints (11) ensure that each customer is visited by an electric vehicle, while constraints (12) allow vehicles to visit a charging station only when required. Constraints (13) guarantee that each vehicle that visits a particular node must also depart from the corresponding node. The arrival and departure times are calculated using constraints (14) and (15), which consider the service and battery charging times. Constraints (16) permit vehicles to visit each node within the corresponding time windows. At the same time, constraints (14)-(16) prevent sub-tours. Constraints (17)–(21) ensure that the delivery and pickup demands of customers are simultaneously met. Finally, constraints (22)-(24) are related to the battery state of charge. For an example instance together with a solution, see Fig. 5.

### 4.1 Literature Review Concerning the EVRP-TW-SPD

In response to mounting environmental concerns and the consequent need for alternative fuel sources in logistics, recent research has focused on developing routing strategies that optimize the transportation of goods while also considering the limited driving range and en-route charging necessities associated with electric vehicles. These problems are commonly referred to as EVRPs or, more broadly, Green Vehicle Routing Problems. Recent, comprehensive surveys of research on EVRPs can be found in (Asghari & Mirzapour Al-e-Hashem, 2021; Moghdani et al., 2021). As the main focus of our work is on the methodology for solving certain types of problems instead of specific problem variants, we refer the interested reader to these survey papers for further reference.

Instead, we point out the differences between the EVRP-TW-SPD with existing problems from the literature. Apart from time window constraints, our problem also considers simultaneous pickup and delivery (SPD) constraints regarding customer deliveries. This practice is commonly associated with reverse logistics. Nevertheless, despite the pivotal role of reverse logistics in promoting sustainability, the number of publications that examine variants of the EVRP-SPD is very limited. So far, only (Yilmaz & Kalayci, 2022) have considered SPD constraints within the scope of EVRPs. Finally, in conventional EVRP models, electric vehicle batteries are assumed to be fully charged upon visiting a charging station. However, our problem considers a more realistic scenario by allowing for partial recharging.

### 4.2 Set-Covering Based ILP Model of the EVRP-TW-SPD

Assignment-type ILP models such as the one presented above for the EVRP-TW-SPD generally do not allow to derive good lower bounds (see, for example, (Angelelli & Mansini,

2002)). In addition, experiments reported in (Akbay et al., 2023) showed that finding any feasible solution to the corresponding model within reasonable execution times for CPLEX becomes difficult, even in the context of small-sized subinstances of the original problem instances.

In a way similar to the one presented for the VSBPP in Sect. 3.3, the EVRP-TW-SPD can be modeled in terms of a set-covering-based ILP in the following way. Let $\mathscr{T}$ be the set of all possible (and feasible) tours, where a tour is defined as the trip of one single vehicle returning to the depot from which it originally left. Each tour $T_r \in \mathscr{T}$ is evaluated by the total distance traveled $d_r$, that is, the sum of the distances of all arcs on the tour. Finally, let $\mathscr{T}_i \subset \mathscr{T}$ be the set of tours that serve customer $i \in V$. With these definitions, the set-covering-based ILP model for the EVRP-TW-SPD, henceforth denoted as $\texttt{ILP}^{\text{EVRP}}_{\text{setcov}}$, can be stated as follows.

$$\textbf{min} \sum_{T_r \in \mathscr{T}} d_r x_r + M \sum_{T_r \in \mathscr{T}} x_r \tag{26}$$

$$\textbf{s.t.} \quad \sum_{T_r \in \mathscr{T}_i} x_r \geq 1 \quad \forall\, i \in V \tag{27}$$

$$x_r \in \{0, 1\} \quad \forall\, T_r \in \mathscr{T} \tag{28}$$

The objective function minimizes the total travel and vehicle costs and constraints (27) ensure that each customer is visited at least once. Note that the set-covering-based formulation is generally used as a post-optimization method in the VRP literature. In (Rochat & Taillard, 1995), following the termination of their algorithm, the authors suggest aggregating all tours from the solutions generated by their algorithm into a set. They then attempt to find an even better solution by resolving the set-covering model based on the tours contained in this set. In contrast, our results will show that CMSA provides a suitable algorithmic framework for iteratively applying heuristics and exact components.

### 4.3 Application of standard Adapt-CMSA to the EVRP-TW-SPD

In the case of the EVRP-TW-SPD problem, we apply the Adapt-CMSA variant from (Akbay et al., 2022). In particular, we first develop an Adapt-CMSA version based on the assignment-type ILP model—that is, model $\texttt{ILP}^{\text{EVRP}}_{\text{std}}$—to the EVRP-TW-SPD. This version of Adapt-CMSA is henceforth labeled Adapt-CMSA- STD. In the context of Adapt-CMSA- STD, the complete set of solution components consists of a component $c_{ij}$ for each arc $a_{ij}$ from $A = \{(i, j) | i, j \in V'_{0,N+1}, i \neq j\}$. Consider the following example. The vector $\mathbf{I}$ comprises all the node indexes for a small problem instance involving three charging stations and five customers. Nodes indexed with 0 and 6 denote the depot.

$$\mathbf{I} = (\underbrace{0,}_{\text{depot}} \underbrace{1, 2, 3, 4, 5,}_{\text{customers}} \underbrace{6,}_{\text{depot}} \underbrace{7, 8, 9}_{\text{charging stations}})$$

Now consider a solution consisting of two tours $T_1$ and $T_2$, where $T_1 =< 0 \to 9 \to 1 \to 4 \to 6 >$ and $T_2 =< 0 \to 2 \to 8 \to 3 \to 7 \to 5 \to 6 >$. In the context of Adapt-CMSA- STD, this solution is represented by $S = \{c_{0,9}, c_{9,1}, c_{1,4}, c_{4,6}, c_{0,2}, c_{2,8}, c_{8,3}, c_{3,7}, c_{7,5}, c_{5,6}\}$, that is, a solution $S$ in Adapt-CMSA- STD is kept in terms of the list of solution components representing the arcs used in any of the tours of $S$.

**Algorithm 3** Pseudo-code of Adapt-CMSA for the EVRP-TW-SPD

1: **input 1:** values for CMSA parameters $t_{\text{prop}}$, $t_{\text{ILP}}$
2: **input 2:** values for solution construction parameters $\alpha^{\text{LB}}$, $\alpha^{\text{UB}}$, $\alpha_{\text{red}}$
3: $S^{\text{bsf}}$ := GenerateGreedySolution()
4: $\alpha_{\text{bsf}} := \alpha^{\text{UB}}$
5: Initialize($n_{\text{a}}, l_{\text{size}}$)
6: **while** CPU time limit not reached **do**
7: $C' := S^{\text{bsf}}$
8: **for** $i := 1, \ldots, n_{\text{a}}$ **do**
9: $S :=$ ProbabilisticSolutionConstruction($S^{\text{bsf}}, \alpha_{\text{bsf}}, l_{size}$)
10: LocalSearch1($S$)
11: **for** all $c \in S$ **and** $c \notin C'$ **do** $C' := C' \cup \{c\}$ **end for**
12: **end for**
13: $(S^{\text{ILP}}, t_{\text{solve}})$ := SolveSubinstance($C'$, $t_{\text{ILP}}$) {This function returns two objects: (1) the obtained solution ($S^{\text{ILP}}$), (2) the required computation time ($t_{\text{solve}}$)}
14: LocalSearch2($S^{\text{ILP}}$)
15: **if** $t_{\text{solve}} < t_{\text{prop}} \cdot t_{\text{ILP}}$ and $\alpha_{\text{bsf}} > \alpha^{\text{LB}}$ **then** $\alpha_{\text{bsf}} := \alpha_{\text{bsf}} - \alpha_{\text{red}}$ **end if**
16: **if** $f(S^{\text{ILP}}) < f(S^{\text{bsf}})$ **then**
17: $S^{\text{bsf}} := S^{\text{ILP}}$
18: Initialize($n_{\text{a}}, l_{\text{size}}$)
19: **else**
20: **if** $f(S^{\text{ILP}}) > f(S^{\text{bsf}})$ **then**
21: **if** $n_a = n^{\text{init}}$ **then** $\alpha_{\text{bsf}} := \min\{\alpha_{\text{bsf}} + \frac{\alpha_{\text{red}}}{10}, \alpha^{\text{UB}}\}$ **else** Initialize($n_{\text{a}}, l_{\text{size}}$) **end if**
22: **else**
23: Increment($n_{\text{a}}, l_{\text{size}}$)
24: **end if**
25: **end if**
26: **end while**
27: **output:** $S^{\text{bsf}}$

### 4.4 The Adapt-CMSA-STD Algorithm

The pseudo-code presented in Algorithm 3 is common to Adapt-CMSA- STD and Adapt-CMSA- SETCOV. It describes the general algorithmic framework of Adapt-CMSA for the EVRP-TW-SPD. First, a feasible solution is generated by calling function GenerateGreedySolution() to initialize the best-so-far solution $S^{\text{bsf}}$. More precisely, this function applies an insertion heuristic which is further explained in Sect. 4.4.1. Following that, in lines 4 and 5, parameters $\alpha_{\text{bsf}}$, $n_a$, and $l_{\text{size}}$ are given initial values. How these variables are managed within the algorithm will be explained below.

During each iteration of Adapt-CMSA- STD, a subinstance $C'$ of the original problem instance is created. Similar to the solution representation, a subinstance is also a set of solution components, that is, $C' \subseteq C$, where $C'$ is initialized to the best solution found so far ($S^{\text{bsf}}$) at the beginning of each iteration. Subsequently, a probabilistic solution construction process shown in lines 8–12 probabilistically generates $n_a$ solutions using function ProbabilisticSolutionConstruction($S^{\text{bsf}}, \alpha_{\text{bsf}}, l_{size}$). This function takes two additional parameters apart from $S^{\text{bsf}}$. These are the parameter $\alpha_{\text{bsf}}$ ($0 \leq \alpha_{\text{bsf}} < 1$), which biases the creation of new solutions towards the best-so-far solution, and the parameter $l_{size}$, which determines the number of options considered at each solution construction step. Note that higher values of $\alpha_{\text{bsf}}$ lead to an increase of similarity between the constructed solutions and $S^{\text{bsf}}$. On the contrary, a higher value of $l_{size}$ results in the construction of more diverse solutions and, consequently, contributes to forming a larger subinstance.

After constructing a solution $S$ by calling the above-mentioned function in line 9 of Algorithm 3, each tour of $S$ undergoes a local search process, as indicated in line 10. This local search procedure applies well-known intra-route operators such as *relocation*, *swap*, and *two_opt* in sequential order. Moreover, the best-improvement strategy is adopted in the context of the applied operators. The so-called *relocation* operator sequentially extracts each node from its existing position within a route and repositions it at an alternative location inside the same route. On the other hand, the *swap* operator works by interchanging the positions of a pair of selected nodes belonging to the same route. Lastly, the *two_opt* neighborhood explores every feasible combination of choosing two non-adjacent nodes in the same route and then reverses the arrangement of the nodes situated between the chosen pair of nodes.

Upon the application of local search, the so-called *merge step* is executed in line 11 in the same way as in any other CMSA algorithm. After probabilistically constructing $n_a$ solutions and forming the subinstance $C'$, the subinstance is solved by first generating a corresponding ILP model based on model $\texttt{ILP}^{\text{EVRP}}_{\text{std}}$ and then solving the model with a CPU time limit of $t_{\text{ILP}}$ seconds with CPLEX in function SolveSubinstance($C'$, $t_{\text{ILP}}$). In order to generate this model, the following constraints are added to $\texttt{ILP}^{\text{EVRP}}_{\text{std}}$:

$$x_{ij} = 0 \quad \text{for all } c_{ij} \in C \setminus C' \tag{29}$$

In other words, if an arc $a_{ij}$ has not been used in any of the solutions that were merged into $C'$, using this arc is forbidden by fixing the value of $x_{ij}$ to zero. It is important to note that incorporating more constraints into the original ILP reduces the search space of the resulting ILP model, thereby facilitating CPLEX's ability to generate a high-quality solution or even the optimal one for the corresponding subinstance. However, note that—due to the employed CPU time limit for each application of CPLEX—the output of function SolveSubinstance($C'$, $t_{\text{ILP}}$), denoted as $S^{\text{ILP}}$, is not necessarily an optimal solution to the subinstance. In any case, $S^{\text{ILP}}$ is subject to the application of a local search method different from the one described before. In particular, this local search procedure utilizes inter-tour neighborhoods such as *exchange (1,1)* and *shift (1,0)*. The *exchange (1,1)* neighborhood investigates all potential two-customer swaps not part of the same tour, whereas the *shift (1,0)* neighborhood examines every option for removing a customer from its existing tour and placing it at any possible location in other tours. As is done within LocalSearch1($S$), operators used by LocalSearch2($S$) employ the best-improvement search strategy.

The self-adaptive nature of Adapt-CMSA- STD can be found in the dynamical adjustment of the values of parameters $\alpha_{\text{bsf}}$, $n_a$, and $l_{\text{size}}$. The value of the dynamic parameter $\alpha_{\text{bsf}}$ is bounded from below by $\alpha^{\text{LB}}$ and from above by $\alpha^{\text{UB}}$. Both $\alpha^{\text{LB}}$ and $\alpha^{\text{UB}}$ are input parameters of the algorithm. Moreover, the value of a step size parameter $\alpha_{\text{red}}$ is employed for systematically reducing the $\alpha_{\text{bsf}}$'s value, if needed. Initially, the value of $\alpha_{\text{bsf}}$ is set to the highest possible value, $\alpha^{\text{UB}}$, as shown in line 4.[1] If the resulting ILP is solved within a computation time $t_{\text{solve}}$ that is below a proportion $t_{\text{prop}}$ of the maximum possible computation time $t_{\text{ILP}}$—that is, if $t_{\text{solve}} \leq t_{\text{prop}} \cdot t_{\text{ILP}}$—$\alpha_{\text{bsf}}$'s value is reduced by $\alpha_{\text{red}}$, as seen in line 15. The reasoning behind this step is as follows. If the resulting ILP can be easily solved to optimality for the respective subinstance, the search space is too small, owing to a relatively low number of free variables. To increase the number of free variables in the ILP, solutions produced in ProbabilisticSolutionConstruction($S^{\text{bsf}}$, $\alpha_{\text{bsf}}$, $l_{size}$) should differ more from $S^{\text{bsf}}$, which is achievable by lowering the value of $\alpha_{\text{bsf}}$.

[1] Remember that solutions constructed with a high value of $\alpha_{\text{bsf}}$ will be rather similar to the best-so-far solution $S^{\text{bsf}}$.

The adjustment of parameters $n_a$ and $l_{size}$ follows a similar scheme as the one described above. Their initial values are set as follows: $n_a := n^{init}$ and $l_{size} = l^{init}_{size}$, which is done in the Initialize($n_a$, $l_{size}$) function. This function can be called under three distinct circumstances: (1) at the beginning of the algorithm (line 5), (2) when solution $S^{ILP}$ is strictly better than $S^{bsf}$ (line 18), and (3) when solution $S^{ILP}$ is strictly worse than $S^{bsf}$ while $n_a$ concurrently exceeds $n^{init}$ (line 21). On the other hand, when $S^{ILP}$ and the $S^{bsf}$ have the same objective function value, the algorithm has the capacity to create larger subinstances, leading to an increase in the values of the three parameters in Increment($n_a$, $l_{size}$) function. Specifically, $n_a$ is increased by $n^{inc}$, and $l_{size}$ is increased by $l^{inc}_{size}$.

### 4.4.1 Probabilistic Solution Construction

The call of function ProbabilisticSolutionConstruction($S^{bsf}$, $\alpha_{bsf}$, $l_{size}$) invokes the execution of one of two heuristics: either (1) a version of the Clarke & Wright (C&W) savings algorithm (Clarke & Wright, 1964), or (2) a variant of the insertion algorithm. The choice of a heuristic is done uniformly at random. Both heuristics exclusively generate feasible solutions. In the following, both construction algorithms and their variants are described in detail.

**Probabilistic C&W savings algorithm.** Similar to the original C&W approach, our algorithm variant begins by generating a set of direct routes, denoted as $R = \{(0 \rightarrow i \rightarrow (N+1)) \mid i \in V\}$. Next, the algorithm initializes a savings list $L$ consisting of pairs of nodes $(i, j)$, where $i$ and $j$ represent customers and charging stations. The savings value $\sigma_{ij}$ for each pair is computed using the following equation:

$$\sigma_{ij} := d_{0i} + d_{0j} - \lambda d_{ij} + \mu |d_{0i} - d_{0j}| \tag{30}$$

Here, $\lambda$ and $\mu$ are the so-called *route shape* and *asymmetry scaling* parameters, respectively. The route shape parameter $\lambda$ prioritizes the selection of nodes based on their distance from each other (Yellow, 1970); parameter $\mu$, on the other hand, scales the asymmetry between nodes $i$ and $j$ (Paessens, 1988). Well-working values for these parameters are identified through a parameter tuning procedure described in Sect. 4.6.2. It is important to note that $L$ only includes pairs of nodes $(i, j)$ that satisfy two conditions: (1) $i$ and $j$ are part of two different tours, and (2) both $i$ and $j$ must be adjacent to the depot in the tour of which they form part. Moreover, solution construction will not only be influenced by the savings values of node pairs $(i, j)$ but also by the fact whether or not arc $a_{ij}$ appears in the current best-so-far solution $S^{bsf}$. For this purpose, an additional value, $q_{ij}$, is calculated for each entry $(i, j) \in L$:

$$q_{ij} := \begin{cases} (\sigma_{ij} + 1) \cdot \alpha_{bsf} & \text{if } c_{ij} \in S^{bsf} \\ (\sigma_{ij} + 1) \cdot (1 - \alpha_{bsf}) & \text{otherwise} \end{cases} \tag{31}$$

The algorithm performs the following sequence of steps until the savings list $L$ is empty.

1. After computing $q_{ij}$ for all entries in $L$, the list is sorted in non-increasing order with respect to the $q_{ij}$ values, and a reduced list $L_r$ is created, containing the first $l_{size}$ elements of $L$.
2. Next, an entry $(i, j)$ is chosen from $L_r$ with respect to the following probabilities:

$$\mathbf{p}(i, j) := \frac{q_{ij}}{\sum_{(i', j') \in L_r} q_{i'j'}} \quad \forall (i, j) \in L_r \tag{32}$$

Note that the higher the value of $\alpha_{bsf}$, where $0 \leq \alpha^{LB} \leq \alpha_{bsf} \leq \alpha^{UB} \leq 1$, the higher the probability of selecting arcs that are part of the best-so-far solution $S^{bsf}$.

3. Then, the chosen tours corresponding to nodes $i$ and $j$ are merged. The merging process is determined by one of the following four possible cases, depending on the direct connection of nodes $i$ and $j$ to the depot:

   (a) **Case 1:**
   - $T_1 :< 0 \rightarrow i \rightarrow \ldots \rightarrow N+1 >$, $T_2 :< 0 \rightarrow j \rightarrow \ldots \rightarrow N+1 >$
   - Merging: Reverse $T_1$, $\mathsf{rev}(T_1)$, and concatenate with $T_2$
   - Result: $T_m :< 0 \rightarrow \ldots \rightarrow i \rightarrow j \rightarrow \ldots \rightarrow N+1 >$

   (b) **Case 2:**
   - $T_1 :< 0 \rightarrow i \rightarrow \ldots \rightarrow N+1 >$, $T_2 :< 0 \rightarrow \ldots \rightarrow j \rightarrow N+1 >$
   - Merging: Reverse both $T_1$ and $T_2$,$\mathsf{rev}(T_1)$, $\mathsf{rev}(T_2)$, and concatenate
   - Result: $T_m :< 0 \rightarrow \ldots \rightarrow i \rightarrow j \rightarrow \ldots \rightarrow N+1 >$

   (c) **Case 3:**
   - $T_1 :< 0 \rightarrow \ldots \rightarrow i \rightarrow N+1 >$, $T_2 :< 0 \rightarrow j \rightarrow \ldots \rightarrow N+1 >$
   - Merging: Concatenate $T_1$ and $T_2$
   - Result: $T_m :< 0 \rightarrow \ldots \rightarrow i \rightarrow j \rightarrow \ldots \rightarrow N+1 >$

   (d) **Case 4:**
   - $T_1 :< 0 \rightarrow \ldots \rightarrow i \rightarrow N+1 >$, $T_2 :< 0 \rightarrow \ldots \rightarrow j \rightarrow N+1 >$
   - Merging: Reverse $T_2$,$\mathsf{rev}(T_2)$, and concatenate with $T_1$
   - Result: $T_m :< 0 \rightarrow \ldots \rightarrow i \rightarrow j \rightarrow \ldots \rightarrow N+1 >$

   Depending on the positions of nodes $i$ and $j$ in the tour, it may be required to reverse one or both of the tours selected to ensure a direct connection from $i$ to $j$. In such a case, the reversed form of tour $T_1$ is represented by $\mathsf{rev}(T_1)$. Subsequently, the feasibility of the merged tour $T_m$ is checked in terms of vehicle loading capacity and time windows. If the obtained route violates vehicle capacity and/or time window constraints, it is deemed infeasible and eliminated from the savings list. A new candidate is then chosen following the procedure already outlined above. In the event that the merged tour is not feasible due to battery constraints, a charging station is inserted into the tour. Determining the optimal charging station location involves identifying the first node in the tour to which the electric vehicle has arrived with a negative battery level. Then, a charging station is inserted between this node and the previous node. After determining the insertion position, the charging station that results in the minimum amount of increase in the overall tour distance is selected and placed into the predetermined position. If the tour remains infeasible, then the same procedure is applied to the previous arcs. In those cases in which the infeasibility persists even after attempting to insert charging stations, the merged tour is discarded, and the associated nodes are taken out of the savings list. Then, the next candidate, pair of nodes, is selected from the savings list following the procedure described above. The tour merging process is repeatedly executed until the saving list is exhausted. Once the merging phase is complete, some of the previously added charging stations may no longer be necessary. Therefore, redundant charging stations are first identified and then removed from the constructed tours.
4. Finally, the savings list $L$ is updated as described above.

As a last step, the final set of tours is converted into its corresponding set of solution components.

**Probabilistic Insertion Algorithm.** Our second constructive heuristic operates by inserting customers into available tours in a sequential manner until all customers are visited. The first customer to be inserted into the tour is chosen based on the distance from the depot or the latest possible visiting time. In particular, the initial tour is established by inserting the

customer with either the greatest distance from the depot or the earliest deadline. We then produce a cost list that outlines all possible insertion points for each unvisited customer, along with the associated costs. To determine the cost of inserting a customer at a particular point, we use the following equation, which calculates the cost of inserting customer $i$ between nodes $j$ and $k$

$$c(j,i,k) = d_{ji} + d_{ik} - d_{jk} \tag{33}$$

Then, $q_{jik}$ is calculated for each entry $(j,i,k) \in L$ as follows:

$$q_{jik} := \begin{cases} (c(j,i,k)+1) \cdot (1-\alpha_{\text{bsf}})(1-\alpha_{\text{bsf}}) & \text{if } c_{ji} \in S^{\text{bsf}} \text{ and } c_{ik} \in S^{\text{bsf}} \\ (c(j,i,k)+1) \cdot (\alpha_{\text{bsf}})^2 & \text{if } c_{ji} \notin S^{\text{bsf}} \text{ and } c_{ik} \notin S^{\text{bsf}} \\ (c(j,i,k)+1) \cdot \alpha_{\text{bsf}}(1-\alpha_{\text{bsf}}) & \text{otherwise} \end{cases} \tag{34}$$

Subsequently, the selection of an entry $(j,i,k)$ from the generated list is carried out based on the probabilities calculated using Eq. (34). Next, if the remaining load capacity of the vehicle permits, customer $i$ is added in between customers $j$ and $k$. In addition, if this insertion is infeasible in terms of battery restrictions, a charging station is inserted into the tour using the process described in the C&W savings algorithm. In situations where the insertion of a customer results in the vehicle exceeding its load capacity or battery capacity (even after charging station insertion) or causes a time window violation, a new tour is initiated, which includes only the respective customer.

After inserting all of the customers and a complete solution is derived, the obtained set of tours is transformed into the corresponding set $S$ of solution components.

### 4.5 The Adapt-CMSA-SetCov Algorithm

The ILP model for solving subinstances in this variant of Adapt-CMSA is model $\texttt{ILP}^{\text{EVRP}}_{\text{setcov}}$ from Sect. 4.2. In this case, the complete set of solution components $C$ consists of a component $c_r$ for each valid tour $T_r \in \mathscr{T}$ (see Sect. 4.2), that is, $C := \{c_r \mid T_r \in \mathscr{T}\}$. Any subset $S \subset C$ such that each customer $i \in V$ is served by exactly one tour of $S$ is a valid solution to the EVRP-TW-SPD problem instance.

The probabilistic solution construction process in Adapt-CMSA- SetCov works in exactly the same way as in Adapt-CMSA- Std. Just that the solutions returned consist of solution components that directly correspond to tours (instead of arcs as in the case of Adapt-CMSA-Std).

Another difference is—as mentioned above—the ILP model used to solve subinstances. In fact, given a subinstance $C'$, the corresponding ILP model is obtained by replacing each occurrence of $\mathscr{T}$ with $C'$, that is, the model only considers those tours as eligible tours that appear subinstance $C'$. However, before returning the solution, it is checked for duplicate occurrences of customers, that is, all redundant customers are initially identified. Afterward, the benefit of removing each redundant customer, which is directly related to the distance of the respective customer with the adjacent nodes, is computed. Subsequently, redundant customers, beginning with the one with the greatest benefit, are removed until each customer appears in a single tour only.

### 4.6 Computational Experiments

The experiments were conducted on the same machines as the ones for the VSBPP problem, that is, on a cluster of machines equipped with Intel® Xeon® 5670 CPUs having 12 cores of 2.933 GHz and at least 32 GB of RAM. Moreover, the proposed algorithms were implemented in C++. Sub-instances in both Adapt-CMSA variants were solved using CPLEX version 20.1 in one-threaded mode. Furthermore, the ILP models representing complete problem instances were solved using CPLEX version 20.1 in standalone mode.

#### 4.6.1 Generation of the problem instances for EVRP-SPD-TW

The algorithm's performance was evaluated utilizing the EVRP-TW problem instances derived by Schneider et al. (2014) from the classical VRPTW instances by Solomon (1987). This dataset includes a total of 92 instances, consisting of 36 small-sized instances and 56 large-sized instances. Small-sized instances include 5, 10, and 15 customers, while large-sized instances contain 100 customers and 21 charging stations. These instances are organized into three distinct groups based on the spatial distribution of customer locations: clustered instances (marked by the prefix"c"), randomly distributed instances (prefix "r"), and a hybrid of random and clustered distributions (prefix "rc"). Each group further contains two subclasses (type 1, respectively type 2) which differentiate instances based on factors such as time windows, vehicle load, and battery capacity.

Schneider's modifications primarily involved integrating charging stations and adjusting battery capacities to ensure instance feasibility. Specifically, one charging station was positioned at the depot, with the remaining stations distributed randomly, yet in such a way that every customer could be reached using at most two charging stations. The battery capacity was determined as the maximum of (1) the need to travel 60% of the average route length of the best-known solution and (2) twice the battery capacity required to traverse the longest arc from a customer to a charging station. These changes also caused the creation of new time windows, as the original ones from Solomon became infeasible due to added constraints related to charging times.

Since Schneider's instances only provided a single demand type per customer, we adapted these to fit the requirements of our EVRP-SPD-TW model. This adaptation involved separating the combined delivery and pickup demands. We applied the method described by Salhi and Nagy (1999) to calculate a ratio $\rho_i = \min\{\frac{x_i}{y_i}, \frac{y_i}{x_i}\}$ using the Cartesian coordinates $(x_i, y_i)$ of each customer $i \in V$. The delivery demand $q_i$ was then computed by multiplying the original demand $\delta_i$ by $\rho_i$, and the pickup demand $p_i$ was obtained by subtracting $q_i$ from $\delta_i$. These modified instances are available at: https://github.com/manilakbay/EVRP-TW-SPD-Instances, accessed on (25/04/2024).

#### 4.6.2 Parameter tuning

As in the case of the VSBPP problem, we employed the scientific tuning software irace (López-Ibáñez et al., 2016) to derive well-working parameter values for Adapt-CMSA-STD and Adapt-CMSA- SETCOV. The tuning process was conducted using six instances, namely `r107`, `r205`, `rc101`, `rc104`, `rc105`, and `rc205`. The budget of irace—that is, the number of algorithm runs allowed for tuning—was set to 2500, and the time limit per instance was fixed to 900 CPU seconds. Moreover, the precision of irace was fixed to two positions behind the comma for numerical parameters. Table 5 presents a summary of the parameters, their domains, and the final values selected for the experimentation.

**Table 5** Parameters, their domains, and the chosen values as determined by irace

| Parameter | Domain | Adapt-CMSA-STD | Adapt-CMSA-SETCOV | Description |
|---|---|---|---|---|
| $\lambda$ | $[1,2]$ | 1.99 | 1.38 | route shape parameter (C&W alg.) |
| $\mu$ | $[0,1]$ | 0.23 | 0.58 | asymmetry scaling (C&W alg.) |
| $l_{\text{size}}^{\text{init}}$ | $\{3,5,10,15,20,50,100,200\}$ | 100 | 10 | initial list size value |
| $l_{\text{size}}^{\text{inc}}$ | $\{3,5,10,15,20,50,100,200\}$ | 15 | 20 | list size increment |
| $n^{\text{init}}$ | $\{1,3,5,10,50,100,200,300,500\}$ | 1 | 10 | initial nr. of constructed solutions |
| $n^{\text{inc}}$ | $\{1,3,5,10,50,100,200,300,400\}$ | 1 | 50 | increment for the nr. of constr. solutions |
| $t_{\text{ILP}}$ | $\{5,7,10,15,20,25,30,35,40\}$ | 40 | 20 | CPLEX time limit (seconds) |
| $\alpha^{\text{LB}}$ | $[0.6,0.99]$ | 0.92 | 0.75 | lower bound for $\alpha_{\text{bsf}}$ |
| $\alpha^{\text{UB}}$ | $[0.6,0.99]$ | 0.98 | 0.86 | upper bound for $\alpha_{\text{bsf}}$ |
| $\alpha_{\text{red}}$ | $[0.01,0.1]$ | 0.07 | 0.07 | step size reduction for $\alpha_{\text{bsf}}$ |
| $t_{\text{prop}}$ | $[0.1,0.8]$ | 0.17 | 0.23 | control parameter for bias reduction |

It is worth highlighting that the obtained values for $n^{\text{init}}$ and $n^{\text{inc}}$ are significantly smaller in the context of Adapt-CMSA- STD when compared to those for Adapt-CMSA- SETCOV. One possible explanation for this observation is that the ILP model used within Adapt-CMSA- STD makes it difficult for the algorithm to be successful. It seems as if the subinstances are required to be as small as possible so that valid solutions can be generated by CPLEX when solving these subinstances in a restricted time. This explanation is also supported by the obtained values for $t_{\text{ILP}}$. The limit for the running time of CPLEX for solving the ILP models of each iteration is about twice as high in the case of Adapt-CMSA- STD. Contrary to this, the value of the $l_{\text{size}}^{\text{init}}$ parameter determined for Adapt-CMSA- STD is much higher than that determined for Adapt-CMSA- SETCOV. Higher $l_{\text{size}}^{\text{init}}$ may be considered as a diversification mechanism, as compensation for dealing with small subinstances.

### 4.6.3 Numerical results

In this section, we provide a detailed experimental evaluation of the proposed algorithms and study their performance in various scenarios. To gain a better understanding of how they perform in different situations, we tested them on small-sized instances with 5, 10, and 15 customers, as well as larger-sized instances with 100 customers. The numerical results for the small-sized instances can be found in Tables 6, 7 and 8, while the results for the larger-sized instances are presented in Tables 9, 10 and 11.

To assess the effectiveness of the algorithms in handling small problem instances, we compared Adapt-CMSA- STD and Adapt-CMSA- SETCOV with the application of CPLEX to the full-size instances. However, since CPLEX cannot handle larger problem instances, we used our probabilistic Clark & Wright savings algorithm (pC&W) and our probabilistic sequential insertion algorithm (pSI) as benchmarks for those scenarios. We ensured that the parameters for both algorithms were set in the same way as for their application within Adapt-CMSA- STD. Finally, we imposed a computation time limit of 150 CPU seconds for small problem instances and 900 CPU seconds for larger problem instances. Each algorithm was applied 10 times to each problem instance. Note also that, to compute objective function values, we set the cost of each vehicle used in a solution to 1000, that is, $M = 1000$.

The first multi-column of each result table concerning the small problem instances (Tables 6–8) presents the instance names and the value of $cs$, the number of copies of each charging station (dummy charging stations) permitted to be utilized by our algorithms. The value of $cs$ was determined for each problem instance as follows. The corresponding ILP model was solved with $cs = 1$ in the first step. With this setting, each charging station can

**Table 6** Computational results for small-sized instances with 5 customers

| Instance | | CPLEX | | | | Adapt-CMSA- STD | | | | Adapt-CMSA- SETCOV | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | *cs* | *m* | best | time | gap(%) | *m* | best | avg | time | *m* | best | avg | time |
| c101C5 | 2 | 2 | 2257.75 | 0.61 | 0.0 | 2 | 2257.75 | 2257.75 | 0.03 | 2 | 2257.75 | 2257.75 | 0.017 |
| c103C5 | 2 | 1 | 1175.37 | 0.58 | 0.0 | 1 | 1175.37 | 1175.37 | 0.77 | 1 | 1175.37 | 1175.37 | 0.975 |
| c206C5 | 2 | 1 | 1242.56 | 0.82 | 0.0 | 1 | 1242.56 | 1242.56 | 0.04 | 1 | 1242.56 | 1242.56 | 0.006 |
| c208C5 | 1 | 1 | 1158.48 | 0.12 | 0.0 | 1 | 1158.48 | 1158.48 | 0.01 | 1 | 1158.48 | 1158.48 | 0.001 |
| r104C5 | 1 | 2 | 2136.69 | 0.03 | 0.0 | 2 | 2136.69 | 2136.69 | 0.10 | 2 | 2136.69 | 2136.69 | 0.011 |
| r105C5 | 1 | 2 | 2156.08 | 0.04 | 0.0 | 2 | 2156.08 | 2156.08 | 0.01 | 2 | 2156.08 | 2156.08 | 0.001 |
| r202C5 | 1 | 1 | 1128.78 | 0.08 | 0.0 | 1 | 1128.78 | 1128.78 | 12.81 | 1 | 1128.78 | 1128.78 | 0.001 |
| r203C5 | 1 | 1 | 1179.06 | 0.04 | 0.0 | 1 | 1179.06 | 1179.06 | 0.32 | 1 | 1179.06 | 1179.06 | 0.068 |
| rc105C5 | 2 | 2 | 2233.77 | 3.10 | 0.0 | 2 | 2233.77 | 2233.77 | 0.13 | 2 | 2233.77 | 2233.77 | 0.061 |
| rc108C5 | 1 | 2 | 2253.93 | 0.27 | 0.0 | 2 | 2253.93 | 2253.93 | 0.01 | 2 | 2253.93 | 2253.93 | 0.003 |
| rc204C5 | 1 | 1 | 1176.39 | 0.36 | 0.0 | 1 | 1176.39 | 1176.39 | 0.10 | 1 | 1176.39 | 1176.39 | 0.015 |
| rc208C5 | 1 | 1 | 1167.98 | 0.17 | 0.0 | 1 | 1167.98 | 1167.98 | 0.74 | 1 | 1167.98 | 1167.98 | 0.037 |
| average | | 1.42 | 1605.57 | 0.52 | | 1.42 | 1605.57 | 1605.57 | 1.25 | 1.42 | 1605.57 | 1605.57 | 0.100 |

**Table 7** Computational results for small-sized instances with 10 customers

| Instance | | CPLEX | | | | Adapt-CMSA- STD | | | | Adapt-CMSA- SETCOV | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | *cs* | *m* | best | time | gap(%) | *m* | best | avg | time | *m* | best | avg | time |
| c101C10 | 2 | 3 | 3388.25 | 109.23 | 0.0 | 3 | 3388.25 | 3388.25 | 0.40 | 3 | 3388.25 | 3388.55 | 0.464 |
| c104C10 | 1 | 2 | 2273.93 | 3.08 | 0.0 | 2 | 2273.93 | 2273.93 | 0.68 | 2 | 2273.93 | 2273.93 | 41.311 |
| c202C10 | 1 | 1 | 1304.06 | 9.02 | 0.0 | 1 | 1304.06 | 1304.06 | 0.64 | 1 | 1304.06 | 1304.06 | 0.142 |
| c205C10 | 1 | 2 | 2228.28 | 0.12 | 0.0 | 2 | 2228.28 | 2228.28 | 15.40 | 2 | 2228.28 | 2228.28 | 0.047 |
| r102C10 | 1 | 3 | 3249.19 | 0.80 | 0.0 | 3 | 3249.19 | 3249.19 | 25.93 | 3 | 3249.19 | 3249.19 | 0.012 |
| r103C10 | 1 | 2 | 2206.12 | 24.90 | 0.0 | 2 | 2206.12 | 2206.12 | 7.50 | 2 | 2206.12 | 2206.12 | 31.325 |
| r201C10 | 3 | 1 | 1241.51 | 87.11 | 0.0 | 1 | 1241.51 | 1241.51 | 32.32 | 1 | 1241.51 | 1241.51 | 22.727 |
| r203C10 | 1 | 1 | 1218.21 | 2.23 | 0.0 | 1 | 1218.21 | 1218.21 | 20.02 | 1 | 1218.21 | 1218.21 | 38.672 |
| rc102C10 | 1 | 4 | 4423.51 | 0.21 | 0.0 | 4 | 4423.51 | 4423.51 | 0.34 | 4 | 4423.51 | 4423.51 | 0.020 |
| rc108C10 | 1 | 3 | 3345.93 | 5.29 | 0.0 | 3 | 3345.93 | 3345.93 | 20.80 | 3 | 3345.93 | 3345.93 | 0.019 |
| rc201C10 | 3 | 1 | 1412.86* | 32393.00 | 16.6 | 1 | 1412.86 | 1412.86 | 2.80 | 1 | 1412.86 | 1412.86 | 1.325 |
| rc205C10 | 1 | 2 | 2325.98 | 0.18 | 0.0 | 2 | 2325.98 | 2325.98 | 0.19 | 2 | 2325.98 | 2325.98 | 0.635 |
| average | | 2.08 | 2384.82 | 2719.60 | | 2.08 | 2384.82 | 2384.82 | 10.59 | 2.08 | 2384.82 | 2384.84 | 11.392 |

Results obtained with a CPU time limit of 9 h

**Table 8** Computational results for small-sized instances with 15 customers. The better result between Adapt-CMSA- STD and Adapt-CMSA- SETCOV is highlighted in case it improves over the corresponding CPLEX result

| Instance | | CPLEX | | | | Adapt-CMSA-STD | | | | Adapt-CMSA-SETCOV | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | *cs* | *m* | best | time | gap(%) | *m* | best | avg. | time | *m* | best | avg. | time |
| c103C15 | 2 | 3 | 3348.46 | 7183.45 | 7.3 | 3 | 3348.46 | 3348.47 | 68.59 | 3 | 3348.46 | 3348.46 | 5.348 |
| c106C15 | 1 | 3 | 3275.13 | 1.28 | 0.0 | 3 | 3275.13 | 3275.13 | 6.49 | 3 | 3275.13 | 3275.13 | 46.034 |
| c202C15 | 1 | 2 | 2383.62 | 62.26 | 0.0 | 2 | 2383.62 | 2383.62 | 18.12 | 2 | 2383.62 | 2393.39 | 7.497 |
| c208C15 | 1 | 2 | 2300.55 | 4.62 | 0.0 | 2 | 2300.55 | 2300.55 | 1.55 | 2 | 2300.55 | 2300.55 | 12.231 |
| r102C15 | 2 | 5 | 5412.78 | 7183.67 | 20.6 | 5 | 5412.78 | 5412.78 | 2.59 | 5 | 5412.78 | 5412.78 | 0.121 |
| r105C15 | 1 | 4 | 4336.15 | 7.60 | 0.0 | 4 | 4336.15 | 4336.15 | 1.47 | 4 | 4336.15 | 4336.15 | 1.219 |
| r202C15 | 2 | 2 | 2361.51 | 7181.88 | 27.3 | 2 | 2358.00 | 2364.90 | 32.32 | 1 | **1507.32** | **1677.46** | 64.285 |
| r209C15 | 2 | 1 | 1313.24 | 4396.03 | 0.0 | 1 | 1313.24 | 1313.24 | 17.41 | 1 | 1313.24 | 1313.24 | 15.623 |
| rc103C15 | 1 | 4 | 4397.67 | 349.61 | 0.0 | 4 | 4397.67 | 4397.67 | 0.34 | 4 | 4397.67 | 4397.67 | 0.302 |
| rc108C15 | 1 | 3 | 3370.25 | 1170.76 | 0.0 | 3 | 3370.25 | 3370.25 | 58.85 | 3 | 3370.25 | 3370.25 | 0.335 |
| rc202C15 | 2 | 2 | 2394.39 | 859.43 | 0.0 | 2 | 2394.39 | 2394.39 | 0.68 | 2 | 2394.39 | 2394.39 | 27.978 |
| rc204C15 | 2 | 1 | 1403.38 | 7183.65 | 28.7 | 1 | **1382.22** | 1385.72 | 68.09 | 1 | **1382.22** | **1382.55** | 71.869 |
| average | | 2.67 | 3024.76 | 2965.35 | | 2.67 | 3022.71 | 3023.57 | 23.04 | 2.58 | **2951.82** | **2966.83** | 21.070 |

mostly be visited once. Then, the value of *cs* was iteratively incremented until the optimal solution to the ILP model no longer improved with respect to the previous iteration. The final value of *cs* was then set to the value of *cs* in the previous iteration. Unfortunately, this procedure is not feasible in large-sized problem instances because the ILP solver cannot derive an optimal solution in a reasonable time frame. Therefore, we adopted a constant number of five dummy charging stations per instance. Next, the columns with the heading '*m*' show the number of vehicles used in the respective solutions. For algorithms Adapt-CMSA- STD, Adapt-CMSA- SETCOV, pC&W, and pSI, the columns labeled 'best' display the best objective function values among the solutions obtained after ten runs. Additionally, columns with the heading 'avg.' show the average objective function values over the best solutions of each of the 10 runs. Moreover, the 'time' columns show the computation time of CPLEX and the average computation times of both Adapt-CMSA variants to generate the best solutions in each run. All indicated times are in CPU seconds. The time limit for CPLEX was set to two hours. The 'gap(%)' columns provide the percentage difference between the optimal solutions obtained and the best lower bounds CPLEX achieves. It is worth noting that if the gap value is zero, CPLEX has found an optimal solution.

Based on the obtained results, the following observations can be made. For small-sized problem instances, CPLEX optimally solved 31 instances. However, for the remaining 5 instances (`rc201C10`, `c103C15`, `r102C15`, `r202C15`, `rc204C15`), it only provided feasible solutions. It is worth noting that these solutions were the best found within 2 h of computation time, except for `rc201C10`, where we allowed 9 h of running time to derive the presented solution. On the other hand, both versions of Adapt-CMSA found optimal solutions, as proven by CPLEX. In the case of the `r202C15` instance, Adapt-CMSA- STD and Adapt-CMSA- SETCOV were even able to improve over the solution obtained by CPLEX by 0.15% and 36.17%, respectively. Furthermore, Adapt-CMSA- STD and Adapt-CMSA- SETCOV could improve the solution obtained by CPLEX by 1.51% in the case of the `rc204C15` instance. Moreover, both variants of Adapt-CMSA require considerably less computation time than CPLEX. More specifically, while CPLEX found its best solutions on average in 2965.35 s, Adapt-CMSA- STD was able to do so in 23.04 s, and Adapt-CMSA- SETCOV was able to do so in just 21.07 s.

The numerical results for the large-sized instances demonstrate that both variants of Adapt-CMSA are superior to pC&W and pSI in terms of both best-performance and average-performance. Although the average computation time required by Adapt-CMSA- STD and

Adapt-CMSA- SETCOV was higher than that required by pC&W and pSI, this was because pC&W and pSI were not able to improve their best-found solutions any further, while the Adapt-CMSA algorithms were still able to explore the search space in order find even better solutions. It is also worth noting that solutions found by both versions of Adapt-CMSA utilize fewer vehicles than those found by pC&W and pSI.

In addition, the following observations can be made concerning the comparison of the performances of Adapt-CMSA- STD and Adapt-CMSA- SETCOV. First, Adapt-CMSA- SETCOV significantly outperforms Adapt-CMSA- STD in the case of random and random-clustered instances; see below for statistical tests. However, from the results presented in Table 9, it seems difficult to come to a definite conclusion in the context of clustered-type instances. Adapt-CMSA- SETCOV seems to provide a slightly better performance both in terms of best and average results. However, in order to back this claim up in a scientifically well-founded way, we also present critical difference (CD) plots as a statistical tool for assisting in the evaluation of the obtained results. As in the case of the VSBPP problem, we used the `scmamp` tool Calvo and Santafe (2016) in order to generate the CD plots. Remember that, in these plots, each algorithm variant is positioned along the horizontal axis according to its average ranking for the considered subset of problem instances. Algorithm variants whose performances fall below the critical difference threshold, computed with a significance level of 0.05, are considered statistically equivalent, as indicated by the horizontal bars connecting their respective markers. According to Fig. 6, there is not a significant difference between the performance of Adapt-CMSA- SETCOV and Adapt-CMSA- STD on clustered instances, while both outperform the probabilistic implementations of the construction heuristics.

In summary, both variants of Adapt-CMSA show a very satisfactory performance both in the context of small and large problem instances. Moreover, Adapt-CMSA- SETCOV shows superiority over Adapt-CMSA- STD, particularly in the context of random and random-clustered instances. These claims are backed up in a statistical way by means of the graphics in Fig. 6. However, we also observed that the performance of Adapt-CMSA- SETCOV decreases in the context of instances with a long scheduling horizon (`C2* R2*` and `RC2*`), see Fig. 6f. Solutions for those instances include fewer routes and hence more customers per route when compared to the solutions for the instances with short scheduling horizons (`C1* R1*` and `RC1*`).

### 4.6.4 Performance Difference Between the two EVRP-TW-SPD ILP Models

Finally, as in the case of the VSBPP, we want to show why Adapt-CMSA- SETCOV generally outperforms Adapt-CMSA- STD. For this purpose, we again generate subinstances of different sizes, translate them both into models $\texttt{ILP}^{\text{EVRP}}_{\text{std}}$ and $\texttt{ILP}^{\text{EVRP}}_{\text{setcov}}$, and solve them with CPLEX. In particular, we generated 10 subinstances by probabilistically constructing 100, respectively 500, solutions (small problem instance) and 50, respectively 100, solutions (large problem instance) and by merging their solution components in order to obtain subinstances. This was done for the small problem instance `r202C15` with 15 customers and for the large problem instance `c101` with 100 customers. Figure 7 shows radar charts that present the obtained results in the four different cases. Each radar plot provides four different measures, averaged over 10 subinstances: (1) the number of variables in the models of the subinstances (top), (2) the relative MIP gap after termination of CPLEX (right), (3) the computation time required by CPLEX (bottom), and (4) the absolute improvement when comparing the result of solving the subinstance with the best individual solution that was used to generate the subinstance. Note that the time limit for CPLEX was set to 20 CPU seconds in all cases. Note that a model is promising if the improvement is large, and the number of variables, the relative MIP gap

**Table 9** Computational results for large-sized clustered instances. The better result between Adapt-CMSA- STD and Adapt-CMSA- SETCOV is highlighted in case it improves over the best result of pC&W and pSI

| Instance | pC&W | | | | pSI | | | | Adapt-CMSA-STD | | | | Adapt-CMSA-SETCOV | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | m | best | avg. | time | m | best | avg. | time | m | best | avg. | time | m | best | avg. | time |
| c101 | 21 | 22854.30 | 23028.50 | 406.70 | 13 | 14788.00 | 15574.68 | 513.05 | 12 | **13043.40** | **13043.42** | 385.13 | 12 | 13057.80 | 13063.54 | 292.56 |
| c102 | 19 | 20764.20 | 21008.31 | 350.05 | 13 | 14664.70 | 15366.11 | 397.20 | 11 | **12056.80** | **12920.23** | 560.77 | 11 | 12073.10 | 12944.34 | 468.81 |
| c103 | 16 | 17548.30 | 17622.53 | 483.74 | 12 | 13641.10 | 14363.90 | 491.02 | 11 | 12004.70 | 12026.90 | 452.13 | 10 | **11134.90** | **11917.80** | 718.09 |
| c104 | 13 | 14388.30 | 14428.41 | 500.50 | 11 | 12404.70 | 13195.66 | 415.63 | 10 | 10872.80 | 11353.78 | 629.96 | 10 | **10870.70** | **10876.49** | 608.43 |
| c105 | 19 | 20679.90 | 21608.72 | 588.59 | 13 | 14622.90 | 14928.32 | 346.88 | 11 | **12023.80** | 12341.60 | 562.10 | 11 | 12034.10 | **12068.86** | 582.74 |
| c106 | 18 | 19797.20 | 20626.70 | 386.43 | 13 | 14713.90 | 14770.94 | 349.02 | 11 | **12013.10** | 12438.06 | 652.00 | 11 | 12025.70 | **12059.29** | 434.80 |
| c107 | 18 | 19842.00 | 20594.06 | 459.03 | 12 | 13685.10 | 14631.25 | 339.14 | 11 | **12006.40** | **12023.97** | 538.41 | 11 | 12026.70 | 12046.38 | 393.01 |
| c108 | 16 | 17589.20 | 18105.10 | 402.57 | 13 | 14617.00 | 14693.32 | 472.70 | 11 | 11994.70 | 12016.10 | 579.51 | 10 | **11025.80** | **11822.60** | 556.58 |
| c109 | 14 | 15520.10 | 16415.00 | 454.81 | 12 | 13534.10 | 13628.65 | 353.78 | 10 | 11042.20 | 11885.30 | 714.89 | 10 | **10941.00** | **11180.77** | 746.17 |
| c201 | 10 | 11333.60 | 12051.53 | 448.41 | 5 | 6081.90 | 6184.08 | 506.19 | 4 | **4629.95** | **4629.95** | 37.59 | 4 | 4678.37 | 4703.43 | 390.96 |
| c202 | 8 | 9256.17 | 9509.34 | 490.70 | 5 | 6136.20 | 6232.74 | 526.69 | 4 | **4629.95** | **4629.95** | 273.58 | 4 | 4664.26 | 4706.94 | 394.84 |
| c203 | 7 | 8188.19 | 8245.88 | 308.89 | 5 | 6247.22 | 6352.57 | 573.05 | 4 | **4632.27** | **4690.06** | 740.49 | 4 | 4641.45 | 4734.31 | 497.60 |
| c204 | 5 | 6159.23 | 6200.12 | 416.22 | 4 | 5314.79 | 5354.18 | 556.74 | 4 | **4633.08** | **4665.78** | 801.76 | 4 | 4660.64 | 4737.07 | 716.94 |
| c205 | 8 | 9209.37 | 9472.50 | 412.12 | 5 | 6161.46 | 6277.49 | 527.66 | 4 | **4629.95** | **4629.95** | 76.87 | 4 | **4629.95** | **4629.95** | 125.43 |
| c206 | 6 | 7234.69 | 7989.99 | 460.46 | 5 | 6269.37 | 6305.64 | 405.68 | 4 | **4629.95** | **4629.95** | 213.28 | 4 | **4629.95** | **4629.95** | 203.04 |
| c207 | 7 | 8062.15 | 8134.61 | 397.72 | 5 | 6278.60 | 6329.67 | 486.32 | 4 | **4629.95** | **4629.95** | 255.85 | 4 | **4629.95** | 4635.27 | 260.34 |
| c208 | 6 | 7167.66 | 7674.06 | 341.68 | 5 | 6225.26 | 6282.06 | 458.79 | 4 | **4629.95** | **4629.95** | 284.78 | 4 | **4629.95** | **4629.95** | 261.72 |
| average | 12.41 | 13858.50 | 14277.37 | 429.92 | 8.88 | 10316.84 | 10615.96 | 454.09 | 7.65 | 8476.64 | 8657.94 | 456.42 | 7.53 | **8373.78** | **8552.17** | 450.12 |

**Table 10** Computational results for large-sized random instances. The better result between Adapt-CMSA- STD and Adapt-CMSA- SETCOV is highlighted in case it improves over the best result of pC&W and pSI

| Instance | pC&W | | | | pSI | | | | Adapt-CMSA-STD | | | | Adapt-CMSA-SETCOV | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | m | best | avg. | time | m | best | avg. | time | m | best | avg. | time | m | best | avg. | time |
| r101 | 28 | 30091.80 | 30543.42 | 586.01 | 21 | 23068.50 | 23707.34 | 389.94 | 18 | **19633.80** | 19939.79 | 653.66 | 18 | 19640.60 | **19661.15** | 678.64 |
| r102 | 23 | 24946.40 | 25955.63 | 227.18 | 19 | 20905.90 | 21536.14 | 310.36 | 17 | 18470.80 | 19292.16 | 707.40 | 16 | **17474.10** | **17696.35** | 798.71 |
| r103 | 19 | 20733.40 | 20958.09 | 397.28 | 17 | 18714.90 | 18773.25 | 410.8 | 15 | 16296.50 | 17050.75 | 711.23 | 14 | **15280.30** | **15306.17** | 639.83 |
| r104 | 15 | 16457.60 | 16490.35 | 401.43 | 15 | 16583.40 | 16615.25 | 447.08 | 13 | 14141.10 | 14255.53 | 616.85 | 12 | **13084.30** | **13111.31** | 766.15 |
| r105 | 23 | 24903.20 | 25040.80 | 398.35 | 17 | 18882.40 | 19784.06 | 727.33 | 15 | 16389.20 | 17212.83 | 680.24 | 14 | **15471.30** | **16346.10** | 611.98 |
| r106 | 20 | 21799.00 | 22295.65 | 407.22 | 16 | 17824.50 | 18599.83 | 545.85 | 15 | 16292.00 | 16836.67 | 701.75 | 14 | **15314.80** | **15441.68** | 746.22 |
| r107 | 16 | 17579.50 | 18458.44 | 459.73 | 15 | 16617.60 | 16678.79 | 278.26 | 13 | 14168.90 | 15016.67 | 680.99 | 12 | **13140.10** | **13669.50** | 783.82 |
| r108 | 14 | 15467.70 | 15979.98 | 445.48 | 14 | 15514.90 | 15546.85 | 479.26 | 12 | 13079.80 | 13531.30 | 667.03 | 11 | **12073.70** | **12998.17** | 742.49 |
| r109 | 18 | 19684.40 | 19907.84 | 575.13 | 16 | 17690.00 | 17740.59 | 445.28 | 14 | 15237.30 | 15674.51 | 759.64 | 13 | **14220.80** | **14468.12** | 744.57 |
| r110 | 15 | 16512.80 | 17014.28 | 514.35 | 15 | 16540.20 | 16610.01 | 532.43 | 13 | 14170.20 | 14905.73 | 528.09 | 12 | **13114.30** | **13544.76** | 770.70 |
| r111 | 15 | 16528.30 | 17349.54 | 433.79 | 15 | 16595.30 | 16859.04 | 387.64 | 12 | **13144.20** | 14584.19 | 696.71 | 12 | 13148.80 | **13965.98** | 716.12 |
| r112 | 14 | 15452.40 | 15476.60 | 456.40 | 14 | 15498.00 | 16239.69 | 417.67 | 12 | 13155.60 | 14053.56 | 471.58 | 12 | **13044.10** | **13078.65** | 850.31 |
| r201 | 16 | 17540.00 | 17576.79 | 600.20 | 4 | 5786.71 | 5823.23 | 570.9 | 4 | **5192.33** | **5216.92** | 720.44 | 4 | 5276.75 | 5363.04 | 187.38 |
| r202 | 10 | 11394.70 | 12692.40 | 369.71 | 4 | 5504.61 | 5585.48 | 447.33 | 3 | 4250.70 | 5020.88 | 688.65 | 3 | **4193.33** | **4940.22** | 876.26 |
| r203 | 8 | 9207.85 | 9229.98 | 442.55 | 3 | 4390.30 | 4539.07 | 530.06 | 3 | **3942.74** | 4352.52 | 868.05 | 3 | 3985.02 | **4060.18** | 822.50 |
| r204 | 4 | 5031.33 | 5064.99 | 522.23 | 3 | 4194.11 | 4252.43 | 341.83 | 3 | 3820.72 | 3854.31 | 787.19 | 3 | **3793.76** | **3827.81** | 876.38 |
| r205 | 11 | 12426.50 | 13326.29 | 473.80 | 3 | 4584.97 | 4620.48 | 306.67 | 3 | **4055.28** | **4124.64** | 731.94 | 3 | 4065.06 | 4126.45 | 360.47 |
| r206 | 9 | 10252.10 | 10691.23 | 418.76 | 3 | 4419.12 | 4513.97 | 591.64 | 3 | **3978.10** | 4065.05 | 756.40 | 3 | 3991.44 | **4047.16** | 784.07 |
| r207 | 6 | 7124.47 | 7663.63 | 329.29 | 3 | 4245.71 | 4329.11 | 466.8 | 3 | **3878.91** | **3910.07** | 659.85 | 3 | 3881.97 | 3918.29 | 878.20 |
| r208 | 3 | 4104.27 | 4912.11 | 398.70 | 3 | 4185.27 | 4226.41 | 427.82 | 3 | 3791.27 | 3829.39 | 849.73 | 3 | **3732.80** | **3776.20** | 895.79 |
| r209 | 8 | 9304.02 | 9864.70 | 498.81 | 3 | 4413.41 | 4456.23 | 554.27 | 3 | 3975.64 | 4015.90 | 665.83 | 3 | **3933.55** | **3977.24** | 765.83 |
| r210 | 7 | 8234.21 | 9104.97 | 500.99 | 3 | 4415.78 | 4450.61 | 463.92 | 3 | **3920.37** | 3984.78 | 755.82 | 3 | 3926.79 | **3961.42** | 740.36 |
| r211 | 6 | 7083.79 | 7224.86 | 548.55 | 3 | 4204.05 | 4262.28 | 408.61 | 3 | 3814.42 | 3893.38 | 825.95 | 3 | **3824.47** | **3857.62** | 799.45 |
| average | 13.39 | 14863.47 | 15340.11 | 452.43 | 9.96 | 11512.16 | 11728.27 | 455.73 | 8.83 | 9947.82 | 10374.85 | 703.70 | 8.43 | **9548.35** | **9788.85** | 732.01 |

**Table 11** Computational results for large-sized random clustered instances. The better result between Adapt-CMSA- STD and Adapt-CMSA- SETCOV is highlighted in case it improves over the best result of pC&W and pSI

| Instance | pC&W | | | | pSI | | | | Adapt-CMSA-STD | | | | Adapt-CMSA-SETCOV | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | m | best | avg. | time | m | best | avg. | time | m | best | avg. | time | m | best | avg. | time |
| rc101 | 24 | 26367.50 | 26696.36 | 347.88 | 20 | 22358.30 | 22413.20 | 522.92 | 16 | **17667.70** | 18513.67 | 718.11 | 16 | 17696.20 | **17741.63** | 629.24 |
| rc102 | 21 | 23187.80 | 24083.73 | 599.41 | 19 | 21221.00 | 21295.40 | 493.34 | 16 | 17576.80 | 17909.78 | 558.28 | 15 | **16558.20** | **16628.46** | 601.70 |
| rc103 | 18 | 19909.70 | 20476.46 | 470.41 | 17 | 19012.30 | 19070.74 | 397.58 | 14 | 15366.90 | 16245.06 | 764.45 | 13 | **14358.20** | **14999.16** | 691.74 |
| rc104 | 15 | 16733.40 | 16778.44 | 399.26 | 15 | 16853.80 | 16937.69 | 511.23 | 13 | 14270.50 | 14315.17 | 637.05 | 12 | **13222.50** | **13261.65** | 698.43 |
| rc105 | 19 | 21037.60 | 21959.83 | 386.88 | 18 | 20123.80 | 20307.09 | 568.26 | 15 | 16500.90 | 16933.42 | 652.30 | 14 | **15470.70** | **15820.90** | 639.84 |
| rc106 | 19 | 20969.80 | 21133.97 | 553.93 | 18 | 20083.80 | 20127.15 | 443.43 | 14 | 15432.50 | 16069.05 | 632.16 | 13 | **14448.30** | **15249.17** | 578.17 |
| rc107 | 16 | 17764.50 | 17918.18 | 499.62 | 15 | 16903.20 | 17878.81 | 442.24 | 13 | 14313.40 | 14437.31 | 769.62 | 12 | **13276.50** | **13386.51** | 738.62 |
| rc108 | 15 | 16743.00 | 16781.76 | 475.26 | 15 | 16891.30 | 17165.01 | 468.15 | 12 | 13226.00 | 13891.71 | 620.56 | 12 | **13184.70** | **13214.50** | 717.49 |
| rc201 | 15 | 16895.90 | 18148.54 | 319.12 | 5 | 7137.50 | 7191.55 | 359.46 | 4 | **5504.77** | 5819.06 | 703.93 | 4 | 5617.75 | **5786.48** | 322.85 |
| rc202 | 12 | 13635.30 | 13899.58 | 476.49 | 4 | 5871.98 | 5941.11 | 495.09 | 4 | **5324.64** | **5442.41** | 593.36 | 4 | 5436.63 | 5541.80 | 196.92 |
| rc203 | 8 | 9387.70 | 10210.36 | 494.81 | 4 | 5677.24 | 5708.69 | 452.79 | 4 | 5109.88 | 5177.69 | 644.21 | 4 | **5086.44** | **5159.15** | 757.20 |
| rc204 | 5 | 6211.87 | 6441.15 | 433.68 | 4 | 5471.72 | 5545.00 | 540.05 | 3 | 4036.49 | 4525.03 | 745.44 | 3 | **3962.88** | **4252.28** | 899.13 |
| rc205 | 12 | 13603.40 | 13790.39 | 408.58 | 4 | 5776.53 | 5943.19 | 470.09 | 4 | **5260.14** | **5338.50** | 607.45 | 4 | 5285.41 | 5375.54 | 314.93 |
| rc206 | 11 | 12707.00 | 13467.00 | 534.29 | 4 | 5805.89 | 5847.72 | 586.12 | 4 | 5234.55 | 5289.90 | 670.19 | 4 | **5210.57** | **5275.72** | 562.86 |
| rc207 | 8 | 9394.32 | 9782.01 | 394.18 | 4 | 5624.55 | 5648.26 | 567.96 | 3 | **4150.60** | 4930.81 | 694.79 | 3 | 4197.81 | **4650.01** | 864.38 |
| rc208 | 6 | 7218.95 | 7299.79 | 520.74 | 3 | 4482.34 | 5381.34 | 428.75 | 3 | 3977.50 | 4046.09 | 762.57 | 3 | **3920.17** | **4002.79** | 750.41 |
| average | 14.00 | 15735.48 | 16179.22 | 457.16 | 10.56 | 12455.95 | 12650.12 | 484.22 | 8.88 | 10184.58 | 10555.29 | 673.40 | 8.50 | **9808.31** | **10021.61** | 622.74 |

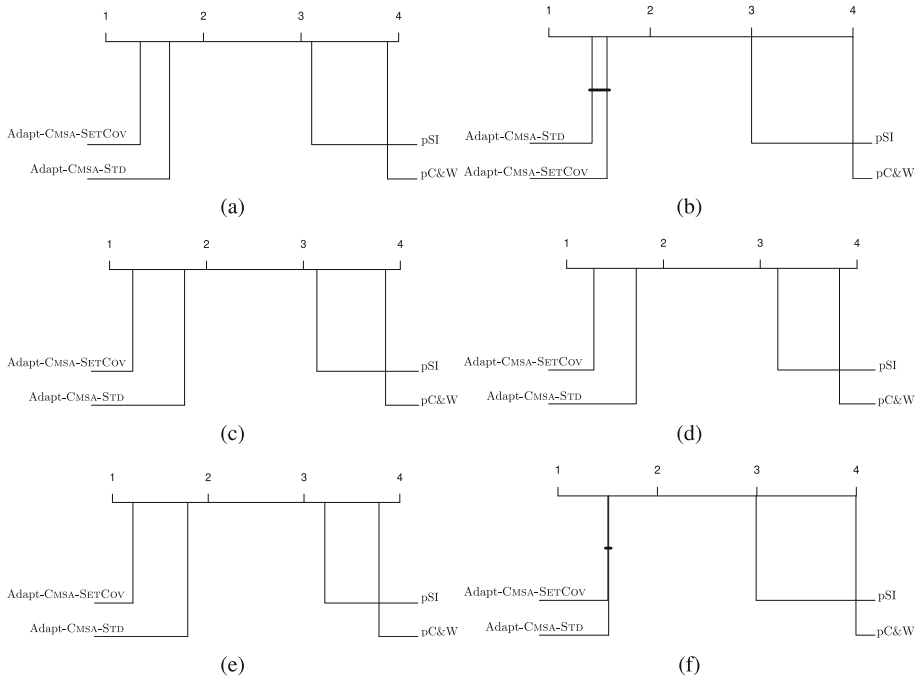
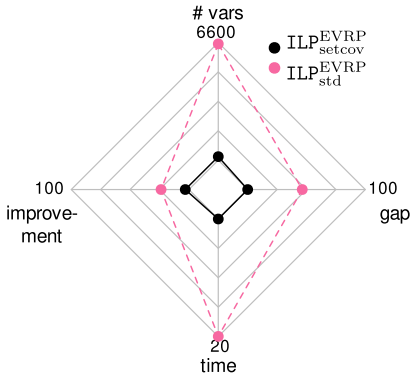



**Fig. 6** Critical difference (CD) plots concerning the results for large instances. The results in (**a**) consider all instances together, while the subsequent plots display the results for subsets of the set of large instances: (**b**) clustered instances; (**c**) random instances; (**d**) random-clustered instances; (e) instances `R1*`, `C1*` and `RC1*`; (f) instances `R2*`, `C2*` and `RC2*`

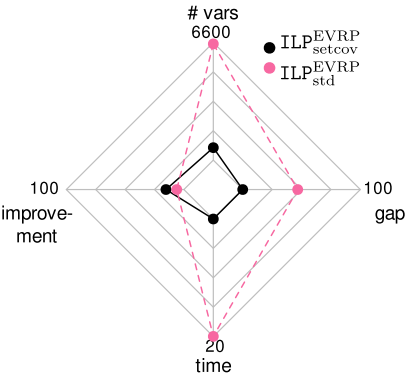
and the required time are low. The radar charts concerning the large problem instance (see Fig. 7c and d) indicate that this is the case for model $\texttt{ILP}^{\text{EVRP}}_{\text{setcov}}$, while the opposite is actually the case for model $\texttt{ILP}^{\text{EVRP}}_{\text{std}}$. Especially the case of the small problem instance considering the lower number of solution constructions (see Fig. 7a) indicates that subinstances must not be too small. Otherwise, there might not be many improvements to be found in the context of model $\texttt{ILP}^{\text{EVRP}}_{\text{setcov}}$.

## 5 Conclusions and future research directions

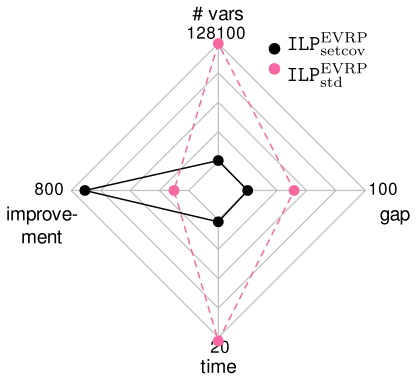
In this paper, we presented the application of different CMSA variants to two NP-hard combinatorial optimization problems. The first problem is known as the variable-sized bin packing problem, and the second one is the electric vehicle routing problem with time windows and simultaneous pickup and delivery. Both considered optimization problems share the property that they can be modeled using an assignment-type integer linear program and a set-covering-based integer linear program. Both models were used to solve subinstances at each iteration of the presented CMSA algorithms. In both cases, the results have shown convincingly that the CMSA variants using the set-covering-based models significantly outperform the CMSA variants using the standard assignment-type models. In fact, in our opinion, CMSA algorithms are an ideal algorithmic framework for taking profit from set-covering-based models for solving optimization problems that can be modeled in this way. This is because CMSA



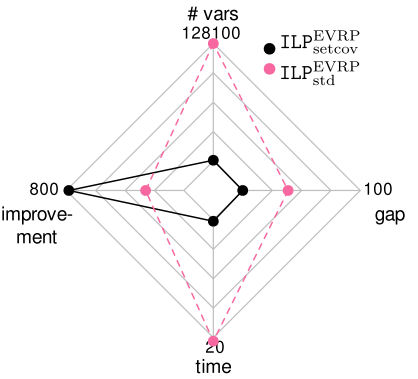

(a) Instance `r202C15` (15 customers), 100 solutions




(b) Instance `r202C15` (15 customers), 500 solutions




(c) Instance `c101` (100 customers), 50 solutions




(d) Instance `c101` (100 customers), 100 solutions

**Fig. 7** Radar charts concerning the comparison of the two ILP models for the EVRP-TW-SPD problem applied to a small problem instance with 15 customers (see (**a**) and (**b**)), and to a large problem instance with 100 customers (see (**c**) and (**d**))

algorithms are less sophisticated and easier to implement in comparison to column generation approaches. In addition, CMSA algorithms can explore search spaces, in contrast to simpler heuristic methods from the literature devised to take profit from set-covering-based models.

When faced with a combinatorial optimization problem that can be modeled by both types of integer linear programming models, we recommend to test the suitability of using the set-covering-based model by performing experiments such as the ones that led to the radar plots in Figs. 4 and 7. They show that it is much more profitable to use the set-covering formulation for solving subinstances because the quality of the solutions found is much higher when compared to the standard integer linear programming model. At the same time, the computational effort is much lower. If this is the case, the use of a set-covering-based model is indicated.

We envisage at least two possible lines for future work. One line consists in consolidating our findings in the context of additional combinatorial optimization problems that can be modeled by set-covering-based models. Another line of work consists of improving the CMSA algorithms presented in this work. In the context of the VSBPP problem, for

example, we only implemented the first solution construction approach that came to mind. Adding additional greedy heuristics for the construction step of CMSA could help to generate potentially different bins that, in combination with other bins, could help to find even better solutions or to improve our results in the few cases in which our algorithm is not able to compete with the state-of-the-art variable neighborhood search technique. Also, in the case of the electric vehicle routing problem, we see potential for improvement by adding additional solution construction techniques.

**Acknowledgements** We acknowledge administrative and technical support by the Spanish National Research Council (CSIC, Spain).

**Author Contributions** Conceptualization, MAA, CB, CBK; methodology, MAA, CB; programming, MAA, CB; writing-original draft, MAA, CB; writing-review and editing, MAA, CB; data curation, MAA, CB; supervision, CB; validation, MAA, CB. All authors have read and agreed to the published version of the manuscript.

**Funding** Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. The research presented in this paper was supported by Grants TED2021-129319B-I00 and PID2022-136787NB-I00 funded by MCIN/AEI/10.13039/501100011033. Additionally, M.A. Akbay and C.B. Kalayci acknowledge support from the Technological Research Council of Turkey (TUBITAK) under Grant no. 119M236. The corresponding author was funded by the Ministry of National Education, Turkey (Scholarship program: YLYS-2019).

**Data availability and materials** For detailed results, interested parties can request information from the corresponding author. The same applies to all the problem instances utilized in this study.

## Declarations

**Conflict of interest** The authors declare no Conflict of interest. The funders played no part in the study's design, data collection, analysis, interpretation, manuscript writing, or decision to publish the results.



## References

Akbay, M.A., Kalayci, C.B., & Blum, C.(2023). Application of Adapt-CMSA to the two-echelon electric vehicle routing problem with simultaneous pickup and deliveries. In: European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar), 16–33. Springer

Akbay, M.A., Kalayci, C.B., & Blum, C.(2023). Application of CMSA to the electric vehicle routing problem with time windows, simultaneous pickup and deliveries, and partial vehicle charging. In: Di Gaspero, L., Festa, P., Nakib, A., Pavone, M. (eds.) Proceedings of MIC 2022 – Metaheuristics International Conference, 1–16. Springer, Cham

Akbay, M. A., López Serrano, A., & Blum, C. (2022). A self-adaptive variant of CMSA: Application to the minimum positive influence dominating set problem. *International Journal of Computational Intelligence Systems, 15*(1), 1–13.

Angelelli, E., & Mansini, R.(2002). The vehicle routing problem with time windows and simultaneous pick-up and delivery. In: Quantitative Approaches to Distribution Logistics and Supply Chain Management, 249–267. Springer, Berlin Heidelberg

Angelelli, E., Mansini, R., & Speranza, M. G. (2010). Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers and Operations Research, 37*(11), 2017–2026.
Applegate, D., Bixby, R., Chvátal, V.,& Cook, W.(1999). Finding tours in the TSP. Technical report, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany
Asghari, M.,& Mirzapour Al-e-Hashem, S.M.J.(2021). Green vehicle routing problem: A state-of-the-art review. International Journal of Production Economics 231, 107899
Bard, J. F., Huang, L., Dror, M., & Jaillet, P. (1998). A branch and cut algorithm for the VRP with satellite facilities. *IIE transactions, 30*(9), 821–834.
Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., & Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research, 46*(3), 316–329.
Blum, C., Pinacho Davidson, P., López-Ibáñez, M., & Lozano, J. A. (2016). Construct, merge, solve & adapt: A new general algorithm for combinatorial optimization. *Computers and Operations Research, 68*, 75–88.
Cacchiani, V., Hemmelmayr, V. C., & Tricoire, F. (2014). A set-covering based heuristic algorithm for the periodic vehicle routing problem. *Discrete Applied Mathematics, 163*, 53–64.
Calvo, B.,& Santafé, G.(2016). scmamp: Statistical comparison of multiple algorithms in multiple problems. The R Journal **8**(1)
Cavaliere, F., Bendotti, E., & Fischetti, M. (2022). An integrated local-search/set-partitioning refinement heuristic for the capacitated vehicle routing problem. *Mathematical Programming Computation, 14*(4), 749–779.
Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research, 12*(4), 568–581.
Cook, W., & Seymour, P. (2003). Tour merging via branch-decomposition. *INFORMS Journal on Computing, 15*(3), 233–248.
Crainic, T. G., Perboli, G., Rei, W., & Tadei, R. (2011). Efficient lower bounds and heuristics for the variable cost and size bin packing problem. *Computers and Operations Research, 38*(11), 1474–1482.
Desrochers, M., & Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science, 23*(1), 1–13.
Djukanović, M., Kartelj, A., & Blum, C. (2023). Self-adaptive CMSA for solving the multidimensional multi-way number partitioning problem. *Expert Systems with Applications, 232*, 120762.
Dupin, N., & Talbi, E.-G. (2021). Matheuristics to optimize refueling and maintenance planning of nuclear power plants. *Journal of Heuristics, 27*(1–2), 63–105.
Ekici, A. (2023). A large neighborhood search algorithm and lower bounds for the variable-sized bin packing problem with conflicts. *European Journal of Operational Research, 308*(3), 1007–1020.
Erdoğan, S., & Miller-Hooks, E. (2012). A green vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review, 48*(1), 100–114.
Ferrer, J., Chicano, F., & Ortega-Toro, J. A. (2021). CMSA algorithm for solving the prioritized pairwise test data generation problem in software product lines. *Journal of Heuristics, 27*, 229–249.
Fleszar, K. (2023). A new MILP model and fast heuristics for the variable-sized bin packing problem with time windows. *Computers and Industrial Engineering, 175*, 108849.
García, S., & Herrera, F. (2008). An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. *Journal of Machine Learning Research, 9*, 2677–2694.
Haouari, M., & Serairi, M. (2009). Heuristics for the variable sized bin-packing problem. *Computers and Operations Research, 36*(10), 2877–2884.
Hemmelmayr, V., Schmid, V., & Blum, C. (2012). Variable neighbourhood search for the variable sized bin packing problem. *Computers and Operations Research, 39*(5), 1097–1108.
Kenny, A., Li, X., & Ernst, A.T. (2018). A merge search algorithm and its application to the constrained pit problem in mining. In: Proceedings of the Genetic and Evolutionary Computation Conference, 316–323
Keskin, M., & Çatay, B. (2016). Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies, 65*, 111–127.
Lamanna, L., Mansini, R., & Zanotti, R. (2022). A two-phase kernel search variant for the multidimensional multiple-choice knapsack problem. *European Journal of Operational Research, 297*(1), 53–65.
López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58.
Machado, A.M., Mauri, G.R., Boeres, M.C.S., & Alvarenga Rosa, R.(2021). A new hybrid matheuristic of GRASP and VNS based on constructive heuristics, set-covering and set-partitioning formulations applied to the capacitated vehicle routing problem. Expert Systems with Applications 184, 115556
Moghdani, R., Salimifard, K., Demir, E., & Benyettou, A. (2021). The green vehicle routing problem: A systematic literature review. *Journal of Cleaner Production, 279*, 123691.
Monaci, M., & Toth, P. (2006). A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing, 18*(1), 71–85.

Nepomuceno, N., Pinheiro, P., & Coelho, A.L.(2007). Tackling the container loading problem: a hybrid approach based on integer linear programming and genetic algorithms. In: Evolutionary Computation in Combinatorial Optimization: 7th European Conference, EvoCOP 2007, Valencia, Spain, April 11-13, 2007. Proceedings, 154–165 . Springer

Paessens, H. (1988). The savings algorithm for the vehicle routing problem. *European Journal of Operational Research, 34*(3), 336–344.

Parker, M., & Ryan, J. (1993). A column generation algorithm for bandwidth packing. *Telecommunication Systems, 2*(1), 185–195.

Pisinger, D.,& Røpke, S.(2010). Large Neighborhood Search. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research and Management Science, vol. 146, 399–419 . Springer US

Rochat, Y., & Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics, 1*(1), 147–167.

Sá Santos, J. V., & Nepomuceno, N. (2022). Computational performance evaluation of column generation and generate-and-solve techniques for the one-dimensional cutting stock problem. *Algorithms, 15*(11), 394.

Salhi, S., & Nagy, G. (1999). A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the Operational Research Society, 50*(10), 1034–1042.

Schneider, M., Stenger, A., & Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science, 48*(4), 500–520.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research, 35*(2), 254–265.

Thiruvady, D., Blum, C., & Ernst, A. T. (2020). Solution merging in matheuristics for resource constrained job scheduling. *Algorithms, 13*(10), 256.

Thiruvady, D., & Lewis, R. (2022). Recombinative approaches for the maximum happy vertices problem. *Swarm and Evolutionary Computation, 75*, 101188.

Yellow, P. (1970). A computational modification to the savings method of vehicle scheduling. *Journal of the Operational Research Society, 21*(2), 281–283.

Yilmaz, Y., & Kalayci, C. B. (2022). Variable neighborhood search algorithms to solve the electric vehicle routing problem with simultaneous pickup and delivery. *Mathematics, 10*(17), 3108.

Zhou, J., & Zhang, P. (2024). Simple heuristics for the rooted max tree coverage problem. In W. Wu & J. Guo (Eds.), *Combinatorial Optimization and Applications* (pp. 252–264). Cham: Springer.