ORIGINAL ARTICLE

Networks

Selecting Fast Algorithms for the Capacitated Vehicle Routing Problem with Machine Learning Techniques.

Roberto Asín-Achá[1*] | Alexis Espinoza[2*] | Olivier Goldschmidt[3*] | Dorit S. Hochbaum[4*] | Isaías I. Huerta[2*]

[1]Department of Informatics, Universidad Técnica Federico Santa María, Chile

[2]Department of Computer Science, Universidad de Concepción, Chile

[3]Riverside County Office of Education, USA

[4]Department of Industrial Engineering and Operations Research, University of California, Berkeley, USA

Correspondence
Roberto Asín-Achá, Department of Computer Science, Universidad Técnica Federico Santa María, Chile
Email: roberto.asin@usm.cl

Funding information
This research was supported in part by AI Institute NSF Award 2112533.

We present machine learning methods for automatically selecting a “best" performing fast algorithm for the Capacitated Vehicle Routing Problem (CVRP) with unit demands. *Algorithm selection* is to automatically choose among a portfolio of algorithms the one that is predicted to work best for a given problem instance, and *algorithm configuration* is to automatically select algorithm’s parameters that are predicted to work best for a given problem instance. We present a framework incorporating both algorithm selection and configuration for a portfolio that includes the automatically configured “Sweep Algorithm", the first generated feasible solution of the Hybrid Genetic Search (HGS) algorithm, and the Clarke & Wright (CW) algorithm. The automatically selected algorithm is shown here to deliver high-quality feasible solutions within very small running times making it highly suitable for real-time applications and for generating initial feasible solutions for global optimization methods for CVRP. These results bode well to the effectiveness of utilizing Machine learning for improving combinatorial optimization methods.

[*] Equally contributing authors.

KEYWORDS
Vehicle routing, machine learning, algorithm selection.

1 | INTRODUCTION

The Capacitated Vehicle Routing Problem (CVRP) is a well-known NP-hard problem that has been extensively studied due to its importance in applications in logistics and transportation. An instance of this problem consists of a depot, where trucks are located, and customer locations and their demands. The goal is to assign customers to trucks and devise tours for trucks, visiting all customers, so that the total demand of the assigned customers does not exceed the trucks' capacity, and so that the sum of the tour lengths is minimum.

Numerous exact and heuristic algorithms have been devised for CVRP. However, no single algorithm dominates the others for all problem instances, either in the case that a solution is required within a short running time, e.g. real-time, or in the case where closeness to optimality is paramount even at the expense of long running times. Our focus here is on algorithms for CVRP that deliver feasible solutions within very short running times.

Algorithm selection is to construct machine-learning-based oracles that are given a portfolio of algorithms, and predict, a priori, which of these algorithms will perform best for a given input instance. The *meta-solver* is the algorithm that queries the automatic algorithm selector and runs the predicted algorithm. We consider here a portfolio of fast algorithms, that includes the well-known heuristic, the Clarke & Wright "savings" algorithm (CW) which is used in one of the most competitive CVRP solvers to date - FILO CVRP solver [3]. A second algorithm in the portfolio is based on the highly competitive Hybrid Genetic Search (HGS) solver [58]. The third algorithm is the automatically configured "Sweep Algorithm", the Meta-Sweep-Algorithm (MSA) [10].

In [10] we introduced a parameterized variant of the Sweep Algorithm. The Sweep Algorithm assigns customers to a truck, in a wedge area of a circle around the depot. The algorithm assigns trucks to serve either "close" or "far" customers where the parameter that controls what is considered "close" or "far" is the *radius parameter*, which is the fraction of the distance from the depot to the farthest customer. As analyzed in [10], the choice of the radius parameter affects the output of the algorithm, and its best value depends on the instance characteristics. This led to the construction of a machine learning algorithm to automatically configure the Sweep Algorithm. That algorithm, the Meta-Sweep-Algorithm (MSA), provides fast and high-quality solutions for CVRP instances and is included in the portfolio of algorithms here.

For the algorithm selector, we compare here the performance of several machine learning methods (Random Forest, k-Nearest Neighbors, Ada Boost, Gradient Boosting) for the purpose of predicting the best-performing algorithm. The machine learning method that delivered the best accuracy is Random Forest (RF) [30]. A unique aspect of the work presented here is the incorporation of automatic algorithm configuration within an automatic algorithm selection framework. Section 3.1 presents this hierarchical framework.

The performance of the automatic selection algorithm, as well as the performance of the automatic configuration algorithm, are measured in terms of several metrics. For the algorithm selection we trained classification models that were evaluated in terms of accuracy. For the algorithm configurator, we used regression models that were evaluated in terms of the Mean Square Error metric (MSE). The resulting meta-solver is evaluated in comparison to two baselines: the single best solver (SBS), i.e. the solver in the portfolio that, on average, performs best, and the virtual best solver (VBS), that is a the solver that performs best for each problem instance. Details and definitions of these performance metrics are given in Section 2.7.

We present here extensive experimental results in which the meta-solver is tested on benchmarks from the CVRP

Library (CVRPLIB)[1]. These are set to use unit demands and the distances are set to be either the L1 (Manhattan) or the L2 (Euclidean) distance norms. For the training dataset, we generate synthetic data sets described in Section 3.2.1.

We note that in [10], the automatically configured Sweep Algorithm, MSA, was originally trained and tested on instances from new benchmarks that are based on real customer addresses in Los Angeles (LA)[2] and New York (NY)[3]. Even though these benchmarks are different from the ones used here, both, for training and testing, the performance of MSA remains robust and of high quality in comparison to the other algorithms in the portfolio.

Our experimental results validate the effectiveness of automatic algorithm selection and configuration. Our hierarchical meta-solver outperforms the Single Best Solver, MSA for both norms instances, by a significant margin, and is able to deliver solutions that are, on average, within 1% from the best obtainable solutions by any of the solvers in the portfolio.

The paper is structured as follows: Section 2 presents the basic concepts and terminology used here as well as relevant related work. Section 3, describes the setup and implementation of our automatically configured Sweep Algorithm, MSA. Experimental results are presented in Section 4. Section 5 includes conclusions and pointers to future research.

2 | PRELIMINARIES AND RELATED WORK

2.1 | The Capacitated Vehicle Routing Problem

Capacitated Vehicle Routing Problem (CVRP) consists of a set of customers with known demands and an unlimited collection of trucks, of given capacity, located at the “depot" which are to deliver the goods to the customers so as to satisfy their demands. A solution to the problem is a partition of the set of customers, assigning to each truck a subset of the customers and a route, that begins and ends in the depot, and visits all assigned customers. Each assigned set of customers’ demand does not exceed the truck’s capacity. The goal is to minimize the sum of the lengths of the tours traversed by all the trucks.

CVRP is a well-known NP-hard problem for which multiple algorithms were devised. In general terms, the proposed algorithms for CVRP can be classified as heuristic, exact, and Machine-learning-based. A recent review of heuristic algorithms is provided in [27]. A recent exact algorithm by [47] was shown to outperform previous exact CVRP approaches. Machine-Learning-based heuristics were recently proposed, for small-size problem instances [13, 4, 28].

2.2 | Clarke & Wright “savings” algorithm

The Clarke & Wright [24] (CW) algorithm, also known as the “savings” algorithm, is a classic heuristic for the CVRP problem that has been extensively used in the literature and as part of different CVRP solvers, like the state-of-the-art FILO solver proposed in [3]. Most of these solvers use the output of CW as a first solution, which they iteratively refine using various stopping criteria. CW is a deterministic algorithm that works as follows:

1. For every pair of customers (i, j), compute the “saving” $s(i, j) = d(D, i) + d(D, j) - d(i, j)$, where $d(a, b)$ corresponds to the distance from customer a to customer b, and D corresponds to the Depot. The “saving” $s(i, j)$ results from including arc i, j in the solution.

[1] http://vrp.galgos.inf.puc-rio.br/index.php/en/

[2] https://data.lacity.org

[3] https://data.cityofnewyork.us

2. Sort the savings list in descending order.
3. Pick next $s(i,j)$ in the sorted list, add arc (i,j) to route r_{t_i} if feasibility is maintained, and if:
   - i and j are not included in a route. In this case, add a new truck to the solution with route $t_{i,j} = \{D, i, j, D\}$.
   - Either i or j, but not both, is included in some route r_{t_i} and i or j are, respectively, the last or first customer in the truck.
   - i is the last customer of one route and j is the first of another route, in which case, both routes can be merged.
4. If there is no $s(i,j)$ left on the list, for each unattended customer i, create route $t_i = \{D, i, D\}$ and return all routes ; otherwise, goto step 3.

As can be seen, for n customers, this procedure requires the computation of all $O(n^2)$ distances between customers. Most of the implementations of CW, in order to reduce this quadratic computation, consider the pair (i,j) only if j is among the k closest customers to i, or i is among the k closest customers to j. *In this paper we use the implementation of the CW algorithm included in the FILO solver [3], using the default setting* $k = 100$.

2.3 | Hybrid Genetic Search algorithm

The Hybrid Genetic Search Algorithm (HGS) for CVRP [57] starts by generating an initial population of feasible and infeasible solutions to the CVRP. To generate a candidate solution one starts from a random permutation of customer indices. This is translated to a feasible solution formed by collection of tours that cover all customers. A process of local search is then used to improve the quality of the solution in terms of a "fitness function" that allows for infeasibility by assigning penalties to violated trucks' capacities. The local search uses operations that include 2-opt, a variant of 2-opt called 2-opt*, relocate, swap, and a variant of swap called swap*. This process is repeated until there is no improvement in the fitness function. At that point, if the resulting solution is feasible, it is added to the population. Otherwise, with probability half the infeasible solution is added to the population, and with probability half the local search continues with adjusted penalties.

Once the initial population is formed, HGS evolves the population over generations using selection, crossover, mutation, and local search operators. The algorithm terminates after a fixed number of generations or when a time limit is reached. The best solution is returned.

Our implementation of the HGS algorithm is to run it until the *first feasible solution* is generated. This is almost always a solution in the initial population. In our experiments that solution was always the first, second or third candidate added to the initial population. In fact, in 95% of the runs it was the first such candidate.

2.4 | The Sweep Algorithm and Our Automatically Configured Implementation

The Sweep Algorithm [34, 25, 32] consists of two phases. In the first phase, customers are allocated to trucks and in the second phase, a short route for each truck is computed, starting from the depot, serving its customers, and returning to the depot. The second phase can use any Travelling Salesperson Problem (TSP) algorithm. In our implementation we solve the TSP using the Lin-Kernigan [41] heuristic, as implemented in the Concorde [8] package, in C language.

The first phase of the Sweep Algorithm, in which customers are allocated to trucks, consists of two sub-phases:

1. Customers are partitioned into two groups. The first group consists of the customers who are within a given distance parameter, **radius**, from the depot. The remaining customers form the second group.
2. In each group:

a. With respect to the depot, compute the polar coordinates of each customer and sort the customers by increasing angle to the depot.

b. One by one, assign the sorted customers such that the total demand of customers assigned to a truck doesn't exceed the truck capacity. Note that customers allocated to a truck in the first group are located in a wedge of the circle centered at the depot, see Figure 1.

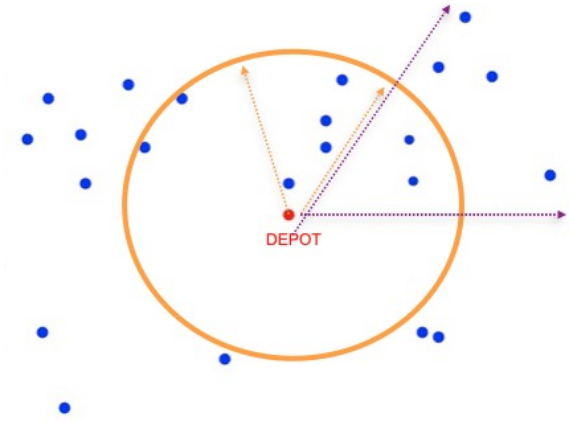
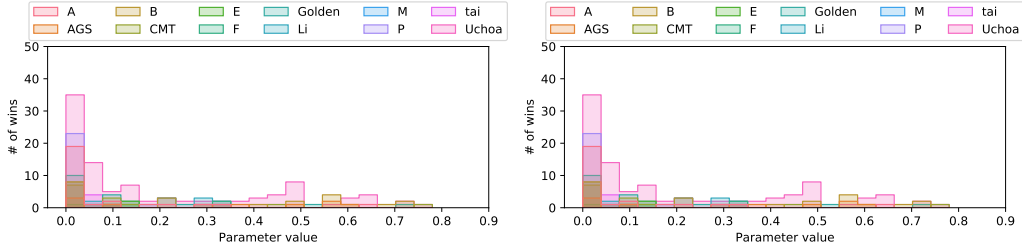


FIGURE 1 For the case of truck capacity 4 and unit demands, this figure illustrates an inner wedge and an outer wedge for the Sweep Algorithm.



(a) Number of wins per each possible parameter value, using L2-norm.

(b) Number of wins per each possible parameter value, using L1-norm.

FIGURE 2 Number of instances out of a total of 262, classified by benchmark, which are best solved per each parameter value, using L1 and L2 distance metrics.

The Sweep Algorithm's parameter: We will use the "radius" parameter r, which is computed as the ratio of the radius of the inner circle divided by the distance to the farthest customer from the depot. This parameter r takes values in the range $[0, 1]$.

Figure 2 shows the number of instances in the CVRPLIB, per benchmark, for which each possible value of r is the best choice. Figure 2a shows the case for Euclidean distances, and Figure 2b, the case where distances are computed using the L1-norm. As can be seen, for all the benchmarks, values close to 0 are highly dominant, which means that trucks serve jointly "close" and "far" customers. Otherwise, the optimal parameter value is hard to predict without the

aid of Machine Learning algorithms.

2.5 | Machine Learning Models for Algorithm Selection

Machine Learning (ML) is an Artificial Intelligence field that is considered a powerful tool for processing and analyzing big data volumes [6]. ML models seek to *l*earn, from given data, called *training set*, a function f that maps an input instance to a corresponding scalar (vector) output, referred to as *label(s)*. ML models can be classified based on how f is learned. When the learning process is based on the use of *ground truth labels*, consisting of a set of input instances and the associated correct output labels (discrete or continuous), then the model is known as **supervised**, e.g. [20]. When labels are not available, the model finds patterns by analyzing the nature of the input data, then the model is said to be **unsupervised**, e.g. [5]. If the learning process of the model is based on both ground-truth data and pattern analysis of the input data, then it is known as **semi-supervised**, e.g. [60]. ML models are also classified based on the output of the mapping f. If the output assumes discrete values, known as "classes", that are used to separate the possible inputs on different categories, the model is said to be a **classification** model. If the output is given as real (floating-point) values, then the model is called a **regression** model.

Automatic algorithm *selection* consists of predicting for a given problem instance and a portfolio of algorithms, which of the algorithms in the portfolio delivers the best quality solution for the instance. Automatic algorithm *configuration* is to predict the value of a given algorithm's parameters that perform best on a given instance. We consider automatic algorithm selection and configuration with supervised machine learning.

The supervised ML algorithms used here are both classification and regression models.

Random Forest (RF), [18], is an ensemble method [17] that constructs a parameterized (n_estimators) number of decision trees [48], that are trained, each one, with a different subset of instances belonging to the training set. To compute the output label, each of the trees "proposes" a result. The final result is determined by a consensus scheme that varies depending if the model is a regression or a classification one. In the regression case, the consensus corresponds to the average of all the decision trees' outputs, while for the classification version, the final output corresponds to the label that repeats (is voted) the most, among the trees' outputs.

k**-Nearest Neighbors (KNN), [30],** is a ML algorithm that bases the output label on the labels of the k closest training examples to the input point we want to label. Although several distance metrics can be used, the Euclidean distance between feature vectors is the most common one. For classification tasks, KNN assigns the output label as the most repeated label among the k neighbors. In the case of a regression task, the label corresponds to the average of the labels of the k neighbors.

Ada Boost (AB), [50], belongs to a class of ensemble methods known as boosting [49]. The ensemble is composed of several ML models of the same type (e.g. decision trees, KNN). In principle, the type of ML model, usually referred to as the "base model", is any ML method that supports weighting the samples. The technique consists of, iteratively, refining the weakest model in the ensemble (i.e. the worst-performing model). First, this weakest model is trained with the complete training set, then a higher weight is assigned to the data points which contribute the, say 10%, largest deviations of the metric being optimized. The weights are computed using a parameter called learning_rate. The new weights model is then added to the ensemble. This is repeated for a number of iterations which is given as a parameter (n_estimators).

Gradient Boosting (GB), [31], combines multiple ML models of the same type (e.g. decision trees, KNN). Gradient Boosting works by starting with an initial model, and computing for that model the gradient of the loss function associated with the predictions. A new model is then trained to reduce errors using a gradient descent-like

approach during training. The new model is added to the ensemble with a weight generated according to the parameter learning_rate. The process is then repeated for predictions computed as the weighted average across the ensemble. The process ends when a stopping rule criterion is met, such as maximum number of iterations, a given value for accuracy, or a convergence threshold. Other than the learning_rate Gradient Boosting uses the parameter n_estimators, the number of new models. The final ensemble is then composed of a set of models where each one contributes to the overall prediction based on its weight.

2.6 | Related Work on Automatic Algorithm Selection and Configuration

In the last few years multiple studies were introduced for algorithm selection and configuration in combinatorial optimization problems. These include research on timetabling [38, 51, 15, 16, 59], MaxSAT [7], scheduling [45, 55], Traveling Salesperson Problem [38, 51, 15, 16, 59, 35], and Multi-Agent Path-Finding [19, 52]. The growing interest in algorithm selection systems motivated algorithm selection competitions [43] and the maintenance of an algorithm selection library ASLib [12].

For CVRP, we are aware of only one work on algorithm selection described in [28]. There, the authors present a Convolutional-Neural-Network-based meta-solver that predicts, from a portfolio of four heuristic algorithms, the one algorithm that performs best. The four algorithms in the portfolio were: the Clarke & Wright (CW) heuristic [24], implemented as described in [39], the Sweep Algorithm [32], implemented as described in [46], an evolutionary algorithm implemented to work with GPUs [1], and a Self-Organizing Map (SOM) algorithm [53]. The prediction is reported to be successful for 79% of the test instances. The authors also report that the CW heuristic is the single best solver, SBS, for 49.8% of the instances. The instances in [28] were generated using a modification of the instance generator proposed in [56], with number of customers in the range $[100, 1000]$. Even though this work includes two of the algorithms used in this paper, the overall setup is considerably different: Firstly, in [28] the training and testing is done on the same set of instances, whereas here the training instances are randomly generated but the testing sets are public benchmarks from the CVRP Library. Secondly, the implementations of the CW and Sweep Algorithms are less advanced than the implementations used here. In particular, the Sweep Algorithm is not configured, as is the case here. Lastly, the execution time of the algorithms is not a factor in [28], with the genetic algorithm reported to take many hours of runtime, whereas here we focus on very fast algorithms that produce a solution within a few CPU seconds.

Additional research on algorithm selection and configuration is surveyed in [36] including for the satisfiability problem, SAT, as well as continuous optimization problems. In this survey there is a proposed taxonomy of algorithm selection and configuration methods indicating that algorithm configuration generalizes algorithm selection, and in that sense is harder. The taxonomy differentiates between “per-set” and “per-instance” algorithm selection/configuration. The “per-set" selection or configuration aims to predict an algorithm from a given portfolio that performs best, on average, for a given set of instances, see e.g. [44]. The “per-instance" definition coincides with the “per-set" definition when the set of instances each consists of a single instance. In this study, we only refer to “per-instance" selection and configuration.

2.7 | Performance Metrics

For evaluating the performance of different regression ML models on the quality of the predicted parameter r for MSA, we use the *Mean Squared Error* [21] (MSE). This measures the average of the squares of the differences between the predicted performance values associated with the different values of r and the best possible performance value

associated with the best choice of r. This best value is determined for each instance by estimating the performance of all possible values of r and picking the one which gives the lowest value of the length of the tours. This best value is referred to as the *ground-truth* label of the instance.

The algorithm selection model is a classification problem. Here we use *accuracy* defined as the ratio between the number of correct labels predicted by the model and the total number of testing instances.

To evaluate algorithm selection methods, two baselines are frequently used:

Single Best Solver (SBS): relates to the performance of the solver with the best average behavior across all the instances used for the first phase (i.e. the best solver on average for the training set).

Virtual Best Solver (VBS): relates to a virtual solver that delivers a perfect selection of the best-performing algorithm for each individual instance.

In this context, to measure the performance of a given algorithm a, we consider a set of instances I and a portfolio of algorithms A. For each algorithm in the portfolio, we compute an evaluation metric m_a that measures the average performance of a across all instances $i \in I$. This is $m_a = \frac{\sum_{i \in I} m_a(i)}{|I|}$, where $m_a(i)$ is the performance of a on individual instance i. For CVRP, we define $m_a(i)$ as the objective value delivered by a on i ($objective_a(i)$) divided by the best objective value delivered by any of the algorithms in the portfolio for i (i.e. the objective value of the VBS), $m_a(i) = \frac{objective_a(i)}{objective_{VBS}(i)}$. This quantity can be viewed as the average approximation factor of algorithm a in comparison to the VBS, which is always ≥ 1. Note that the value of this metric, for VBS, is equal to 1, $m_{VBS} = 1$, which is the best value possible.

We adopt here the performance measure $\hat{m}$ introduced in [43] for the purpose of evaluating the performance of an algorithm selection system *s*. In our case we study the performance of three algorithm selection systems: automatic algorithm selection, automatic algorithm configuration, or a hierarchical combination of both automatic selection and configuration. The performance of a system is expected to improve on the performance of SBS while approximating as closely as possible the performance of VBS. To that end, $\hat{m}_s$ for a system s is defined as follows:

$$\hat{m}_s = \frac{m_s - m_{VBS}}{m_{SBS} - m_{VBS}}, \tag{1}$$

where m_s is computed with the objective values delivered for each instance by the system, e.g. the meta-solver. Values of $\hat{m}_s = 0$ mean that the selector/configurator performs as well as the VBS and $\hat{m}_s = 1$ means that it performs as well as the SBS. Greater than 1 values of $\hat{m}_s$ mean that the use of the selector/configurator is worse than using a single algorithm/configuration, SBS, for all the instances in the test. The closer to zero the value of $\hat{m}_s$ the better is the performance of the system.

3 | ALGORITHM SELECTION AND CONFIGURATION FOR CVRP

3.1 | Algorithm Selection-Configuration Framework

We propose here a hierarchical framework to create a meta-solver composed of both algorithm selection and configuration. Figure 3 provides a schematic illustration of this framework. A meta-solver based on this framework receives as input an instance of a problem. It computes a characterization of that instance that is then fed into a trained algorithm selector that predicts which algorithm in the portfolio of p algorithms would give the best possible solution. A, possibly modified, characterization of the instance is then fed into the chosen algorithm's trained configurator which recommends a most promising set of parameters' values for the given instance. In our case $p = 3$ and only MSA

requires a single parameter configuration, r.

An alternative to this framework would be to merge in a single ML Oracle the task of choosing both, the most promising algorithm and its parameters, considering all possible set of parameters of each of the candidate algorithms as one distinct algorithm. However, we do not use this approach due to its drawbacks: First, such a model would have to learn and predict from a huge number of, likely unbalanced, classes which is much more challenging for a Machine Learning algorithm than dealing with a smaller number of classes. Second, with modifications such as adding an algorithm to the portfolio, or adding parameters to the algorithms, one would have to train the model from scratch, whereas the hierarchical approach used here would only require adjusting the affected modules.

We treat algorithm configuration here as algorithm selection by choosing (100) discrete values for the parameter r in the range $[0 - 1]$ and considering each parameter value in the Sweep Algorithm as a distinct algorithm. This is a setup mentioned in [42] for situations where there is a bounded continuous domain for the parameter, that can be discretized.

The process for building the algorithm selector and configurator is composed of the following tasks:

Training and testing data preparation: For this task, benchmark sets have to be collected, and all possible algorithms/configurations have to be tested against those benchmark sets. For each experiment, the cost obtained by the algorithm and the time of the execution is stored. In [2], some guidelines about this task are given. Details on how we carried out this task for CVRP can be seen in Subsection 3.2.1.

Instance characterization: The instances are characterized by devising and selecting a set of features that are most informative. It is also important that computation of the features of an instance is done efficiently since the focus of algorithm selection is low running time overall. The characterization for CVRP is discussed in Subsection 3.3.

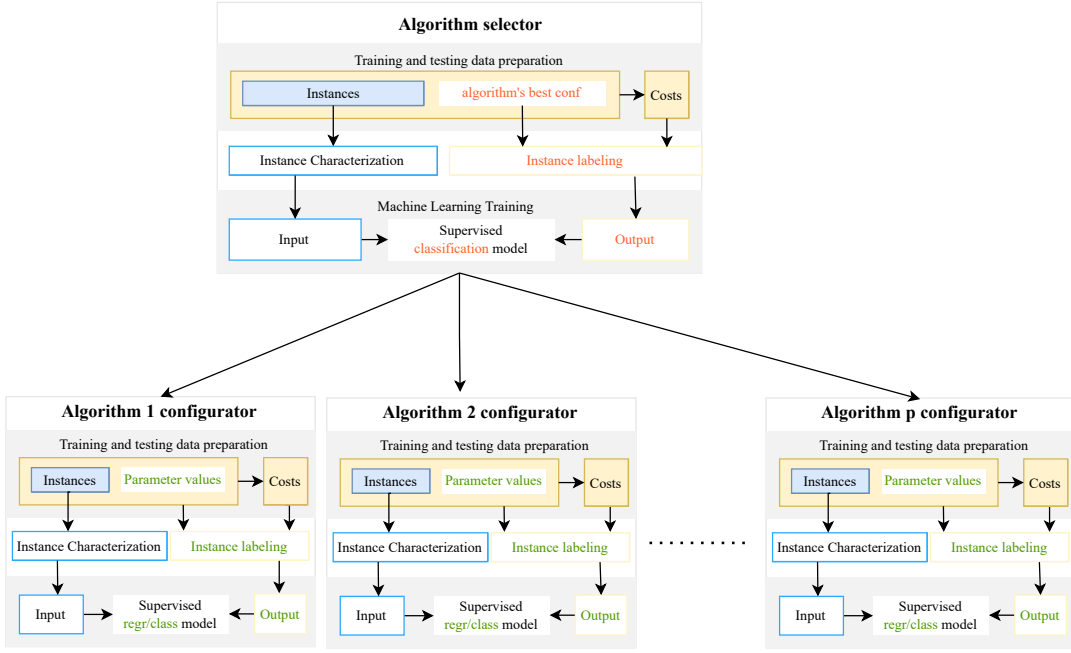


FIGURE 3 Hierarchical algorithm selection and configuration framework.

Instance labeling: Since our approach here is supervised, we need to label our training set. For the algorithm selector, these labels correspond to CW, HGS or MSA, depending if the best-computed solution is found by CW, HGS or MSA, respectively. For MSA's configurator, the labels correspond to the expected normalized cost per each of the 100 possible values of r. Details on this are provided in Subsection 3.4

Machine learning models: A machine learning model is defined by its input, output, and the function that maps the input into the output. The input consists of features determined in the characterization step and the output corresponds to labels. The learning of the mapping is then computed by ML algorithms that adjust the parameters of the underlying structure of the model (neural network, decision trees, etc). After this, the model is validated by comparing the model's output with the labels in the testing data. Our algorithm selector is a classification model since its output corresponds to one of the three possible classes: CW, HGS or MSA. The MSA's configurator is a regression model since its output corresponds to a real value which corresponds to the expected normalized cost for each possible value of r. Details on the algorithms used to train these models are given in Subsection 3.5.

3.2 | Generating CVRP Benchmarks for Training and Testing

As discussed in [2], choosing adequate training and testing sets for building ML models for Combinatorial Optimization problems is crucial for their further applicability and use by Operations Research practitioners. Here we follow a principle used in [35], where the training set corresponds to a synthetically generated data set built with the objective of capturing as much variation in the instances as possible. If such training synthetic data is successfully generated, then the trained models should perform well on public benchmarks that mimic real-world instances.

3.2.1 | Training set

To generate the training set we used TSP instance generators and converted the TSP instances to CVRP instances. The TSP generators used are NETGEN, NETGENM, and TSPGEN, where NETGEN and NETGENM were proposed in [37] and the TSPGEN was proposed in [14]. These generators were successfully used for automatic algorithm selection for TSP problems, [35].

We used a total of 1500 TSP instances: 500 from NETGEN, 500 from NETGENM, and 500 from TSPGEN. The size of the instances varies from 20 to 15360 with the average instance size equal to 2547. Each of these TSP instances are converted into CVRP instances by first choosing three possible positions of the depot: 1) randomly, 2) at the center, and 3) at a randomly chosen corner of the bounding box. Secondly, for each modified instance, four different truck capacities were selected. An alternative way of expressing truck capacity is specifying the minimum number of trucks required to satisfy the total demand. For capacity value q, the minimum number of trucks required is $\lceil \frac{n}{q} \rceil$. The truck capacities were randomly generated in the range $[1, \sqrt{n}]$ following a uniform distribution. These result in a total of 18000 instances in the training set[4].

To understand the profile of the training dataset, we ran the Sweep Algorithm for 100 values of the radius parameter. Table 1 lists the 10 best parameter values and their percentage of wins on the entire training set for the Sweep Algorithm. From the results, reported in Table 1, we determine the best parameter of the Sweep Algorithm for each instance. We call the Sweep Algorithm with the best parameter, the Virtual Best Sweep Algorithm (VSw). We then ran HGS and CW solvers and compared their results to those of the VSw. The percentage of wins of each solver and average running times are reported in Tables 2 and 3 respectively.

[4] The collection of instances in the training set is accessible at `https://drive.google.com/file/d/1wDcuS6qYzf8CSJu2KUyVu26PhQS-pzlK/view?usp=drive_link`.

Parameter Value r L2 norm	0.00	0.01	0.02	0.03	0.04	0.98	0.94	0.95	0.96	0.93	...
% Wins L2 norm	30.43	29.71	27.68	25.50	23.67	23.63	23.58	23.51	23.42	23.40	...
Parameter Value r L1 norm	0.00	0.01	0.02	0.03	0.04	0.98	0.97	0.96	0.92	0.95	...
% Wins L1 norm	29.88	29.21	27.19	25.13	23.28	23.28	23.23	23.21	23.16	23.16	...

TABLE 1 Ranked list of ten best parameter values for the Sweep algorithm in order of percentage of wins for the training set instances with L2 and L1 norms.

Norm	% VSw	% CW	% HGS
L2 norm	52.7	1.3	46.0
L1 norm	51.7	1.0	47.3

TABLE 2 Percentage of best-solved instances per algorithm for training set instances with L2 and L1 norms.

Norm	t_Sw	t_CW	t_HGS
L2 norm	0.504	2.156	169.426
L1 norm	0.451	1.909	167.027

TABLE 3 Average runtime in CPU seconds per algorithm for training set instances with L2 and L1 norms.

The results, in Tables 2 and 3, are shown for L2 and L1-norms. The results reported in Table 2 show the Virtual Best Sweep Algorithm (VSw) is dominant in more than half of the instances for both norms. HGS ranks second with a share of around 45% of the instances, and CW ranks last with fewer than 2% wins. We also observe that the success rate of HGS is slightly higher for L1-norm, than it is for L2-norm.

Concerning the running times, Table 3, shows that HGS is by far the slowest and VSw is faster than CW, taking, approximately, a quarter of the time needed by CW. This is despite running HGS, in most cases, only until the first candidate solution is added to the population. We note here that for several of the largest instances (size greater than 10000), HGS is not able to compute its first solution in a time limit we set to 1 hour. The times reported in Table 3 do not consider such instances.

3.2.2 | Testing set

As specified earlier, our testing set instances are taken from CVRPLIB benchmarks. These include sets A, B, and P from [11], set E from [23], set F from [29], sets M and CMT from [22], Golden from [33], Li from [40], Uchoa from [56], and AGS from [9]. The number of instances in each benchmark A, B, P, E, F, M, CMT, Li, Uchoa, and AGS are $27, 23, 23, 13, 3, 5, 14, 12, 100$ respectively.

The total number of instances in the testing set is 264. The average instance size, excluding AGS, is 263 and the average instance size of AGS is 12200. Since here we consider instances with unitary demand only, we adapted the instances in the CVRPLIB, with customers' demands bigger than one, to unitary demands and, for each instance, we scaled the truck's capacity to a randomly generated value in the range $[1, \sqrt{n}]$, where n is the total number of customers of the instance.

To understand the profile of the testing set, we provide, in Table 4, the percentage of wins of the ten best parameter values for the Sweep Algorithm and their percentage of wins. Table 5 reports the percentage of wins of HGS, CW, and VSw in the test set.

Parameter Value r L2 norm	0.00	0.02	0.01	0.03	0.04	0.05	0.06	0.07	0.08	0.10	...
% Wins L2 norm	54.96	54.58	54.20	52.67	50.38	49.23	46.18	42.27	40.84	36.26	...
Parameter Value r L1 norm	0.02	0	0.01	0.03	0.04	0.05	0.06	0.07	0.08	0.09	...
% Wins L1 norm	53.82	53.44	52.67	51.15	48.09	48.09	45.42	41.60	39.31	34.73	...

TABLE 4 For the testing set, with L2 and L1 norms: List of the ten best parameter values ordered according to the percentage of best-solved instances.

Bench.	% VSw	% CW	% HGS
A	18.52	3.7	**77.78**
AGS	**90.0**	0.0	10.0*
B	39.13	13.04	**47.83**
CMT	**50.0**	0.0	50.0
E	**54.55**	0.0	45.45
F	0.0	0.0	**100.0**
Golden	**90.0**	0.0	10.0
Li	**58.33**	16.67	25.0
M	**80.0**	20.0	0.0
P	**62.5**	0.0	37.5
tai	38.46	15.38	**46.15**
Uchoa	**64.0**	0.0	36.0

(a) Percentage of wins comparison between VSw, CW, and HGS, for L2-norm instances.

Bench.	% VSw	% CW	% HGS
A	**55.56**	3.7	40.74
AGS	**100.0**	0.0	0.0
B	**47.83**	17.39	34.78
CMT	**78.57**	0.0	21.43
E	**63.64**	0.0	36.36
F	**33.33**	33.33	33.33
Golden	**75.0**	25.0	0.0
Li	**50.0**	41.67	8.33
M	**100.0**	0.0	0.0
P	**70.83**	4.17	25.0
tai	23.08	15.38	**61.54**
Uchoa	**68.0**	0.0	32.0

(b) Percentage of wins comparison between VSw, CW, and HGS, for L1-norm instances.

TABLE 5 Percentage of best-solved instances per algorithm per benchmark set. Headings legend: % VSw, % CW, % HGS = percentage of instances in the benchmark best solved by Virtual Best Sweep (VSw), CW, and HGS respectively.

To compare the profiles of the test set versus that of the training set, we note that VSw dominates HGS and CW on most of the benchmarks in the test set. On average, for the L2 instances, VSw dominates on 56.85% of the instances, HGS dominates on 39.69% of the instances, and CW dominates on 3.43% of the instances. For the L1 instances, VSw dominates the other solvers on 64.50% of the instances, HGS dominates on 28.24% of the instances, and CW does so on 7.25% of the instances. In the test set the differences in running times between the three solvers is negligible, since most of the instances can be considered small (average size of 263). This is with the exception of the benchmark set AGS whose instances, on average, have size 12200. Again for most of the instances in the AGS benchmark set, HGS is not able to compute its first solution within a time limit of 1 hour.

3.3 | Instance Characterization and Features

We use features of CVRP instances that are fast-to-extract descriptive statistics. The following features proved to be most successful in preliminary experimentation:

Number of customers: The total number of customers in the CVRP problem.

Capacity: The capacity of the trucks. For this work, we consider uniform capacity across all the trucks, but this can easily be converted to a vector of capacities if this is not the case.

Distance of the depot to the center: The distance between the depot and the center of the map of customers (centroid), divided by the distance of the depot to its farthest customer.

Number of trucks: The minimum number of trucks needed to solve the problem.

k-circular convolution: To capture the distribution of the customers, we take k-concentric circles, centered at the depot, so that the last circle's radius is equal to the distance of the depot to the farthest customer, r_{max}. The radii of the other $k-1$ circles r_i, $i = 1...k-1$, are equal to $\frac{i}{k} \cdot r_{max}$. For each circle $i > 0$, we compute the percentage of customers that lie in the ring between circles i and $i-1$. We choose here $k = 10$.

3.4 | Instance Labeling

We used the results generated for the training set to create labels for the instances. For the algorithm selector, the labels indicate which of the three algorithms provides the best value, HGS, CW, or VSw. In case of ties, the tie is broken in favor of the faster algorithm.

For the algorithm configurator, we label each instance i with a 100-array of real values each of which is the normalized objective value $\text{norm\_objective}(i, r)$, defined below, which is a value in $[0, 1]$. The definition of the normalized objective for instance i uses the following notation: Let $\text{objective}(i, r)$ be the objective value delivered by the Sweep Algorithm with parameter value r; $\text{min\_objective}(i)$ be the best/smallest objective value among all values of r; and $\text{max\_objective}(i)$ be the worst/largest objective value among all values of r. With this notation the normalized objective is defined as:

$$\text{norm\_objective}(i, r) = \frac{\text{objective}(i, r) - \text{min\_objective}(i)}{\text{max\_objective}(i) - \text{min\_objective}(i)}$$

3.5 | Comparing the Performance of the Machine Learning Models

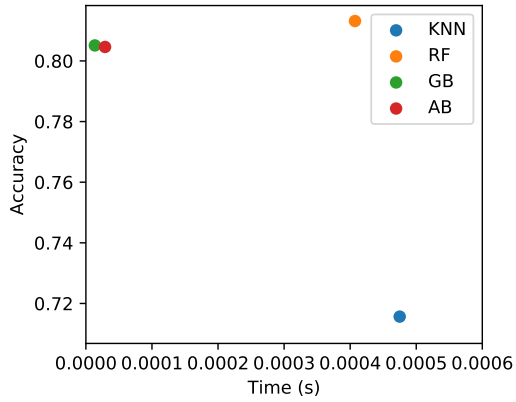


FIGURE 4 Accuracy and Time plot of Classification Machine Learning algorithms trained for the algorithm selector.

To select a suitable machine learning model, we compared the performance of the four machine learning models, discussed in Section 2.5: k-Nearest Neighbors, Random Forest, Gradient Boost, and Ada Boost on the training set. The implementation used for these models is the one in the Scikit-learn modules of Python. The parameters of the

models were adjusted by performing a grid search, using k-fold cross validation [54], with $k = 5$.

The training and testing of the four ML algorithms was done on a computer with an Intel Xeon Haswell processor@2.3Ghz and 64GB of RAM. The performance of the models is compared and reported here for the L2-norm instances of the training set. For Ada Boost and Gradient Boost, we use decision trees as base models. The performance on the L1-norm instances is very similar.

For the algorithm selector, we trained a classification model using the labels described in subsection 3.4. The performance of the machine learning models is measured in terms of accuracy (bigger is better), and average running time. As shown in Figure 4 Random Forest (RF) provides the highest accuracy. Although GB is the fastest method, all running times are negligible, whereas the improvement in the accuracy of RF is significant. The selected parameters of RF are: criterion = gini, n_estimators = 200.

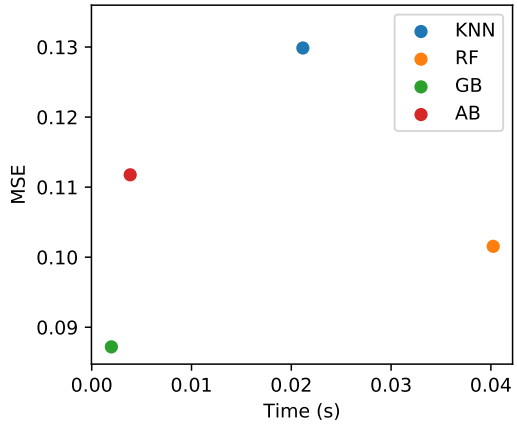


FIGURE 5 MSE and Time plot of Regression Machine Learning algorithms trained for the MSA configurator.

For the algorithm configurator, we trained a regression model that outputs a 100-array of real values as described in section 3.4. The performance of the four ML models is measured in terms of Mean Square Error (MSE) (lower is better), and average running time. As seen in Figure 5, Gradient Boosting (GB) has the best quality MSE and fastest running time. Hence we choose GB for the algorithm configurator. The selected parameters of GB are: loss = squared_error, learning_rate = 0.1, n_estimators = 100.

4 | EXPERIMENTAL RESULTS

4.1 | The Effectiveness of the Algorithm Configurator

To assess the effectiveness of the algorithm configurator, we first compare the performance of the automatically configured Sweep Algorithm (MSA) with the performance of the other algorithms in the portfolio. This comparison is shown in Table 6 for the L2 and L1-norm test instances. The results in the table demonstrate that MSA and HGS are highly dominant as compared to CW, for most benchmarks, for both norms. Overall, MSA finds the best solutions for 45.8% of the L2-norm instances and 55.3% of the L1-norm instances. HGS finds the best solutions for 50% of the L2-norm instances and 34% of the L1-norm instances. Finally, CW finds the best solutions for 4.2% of the L2-norm

instances and 10.7% of the L1-norm instances. The sizes of all testing instances, with the exception of AGS, are quite small. The slower running time performance of HGS is therefore clearly manifested only for that benchmark in the testing set. However, for the training instances which are much larger, with 2547 customers on average, HGS is slower by a factor of 100 or more.

Next we compare MSA, the output of the algorithm configurator, with the virtual best Sweep Algorithm, VSw. For this we contrast the results shown in Table 6 with those presented in Table 5, Section 3.2.2. This comparison indicates that the percentage of wins of MSA, is reduced compared to the percentage of wins of VSw. The difference, in favor of VSw is, 11% for the L2-norm, and 9% for the L1-norm.

In terms of the quality of the configurator's prediction, we trace the errors in the prediction to two sources, both related to the dominant performance of the value $r = 0$. When the ground truth value is 0, our configurator often predicts other values, though mostly values of $r \leq 0.3$. A second source of errors originates in predicting $r = 0$ even though the ground truth is other values.

Bench.	% MSA	t_MSA	% CW	t_CW	% HGS	t_HGS
A	18.52	0.004	3.7	0.005	**77.78**	0.01
AGS	**80.0**	1.988	0.0	20.586	20.0	139.258
B	39.13	0.007	13.04	0.006	**47.83**	0.01
CMT	**50.0**	0.009	0.0	0.007	**50.0**	0.03
E	45.45	0.004	0.0	0.006	**54.55**	0.02
F	0.0	0.01	0.0	0.006	**100.0**	0.02
Golden	**65.0**	0.03	0.0	0.019	35.0	0.21
Li	8.33	0.121	25.0	0.068	**66.67**	0.136
M	**80.0**	0.015	20.0	0.009	0.0	0.06
P	**58.33**	0.003	0.0	0.005	41.67	0.01
tai	23.08	0.023	23.08	0.008	**53.85**	0.05
Uchoa	**51.0**	0.094	0.0	0.03	49.0	0.45

(a) Percentage of wins and average runtime comparison between MSA, CW, and HGS, for L2-norm instances.

Bench.	% MSA	t_MSA	% CW	t_CW	% HGS	t_HGS
A	**55.56**	0.004	3.7	0.006	40.74	0.01
AGS	**90.0**	1.965	0.0	18.674	10.0	159.834
B	**39.13**	0.007	26.09	0.007	34.78	0.01
CMT	**71.43**	0.01	0.0	0.007	28.57	0.03
E	**63.64**	0.004	0.0	0.006	36.36	0.01
F	0.0	0.01	**66.67**	0.007	33.33	0.02
Golden	**60.0**	0.044	30.0	0.019	10.0	0.22
Li	0.0	0.117	**83.33**	0.064	16.67	0.126
M	**80.0**	0.016	0.0	0.01	20.0	0.05
P	**70.83**	0.003	4.17	0.006	25.0	0.01
tai	23.08	0.02	15.38	0.008	**61.54**	0.04
Uchoa	**59.0**	0.082	0.0	0.035	41.0	0.01

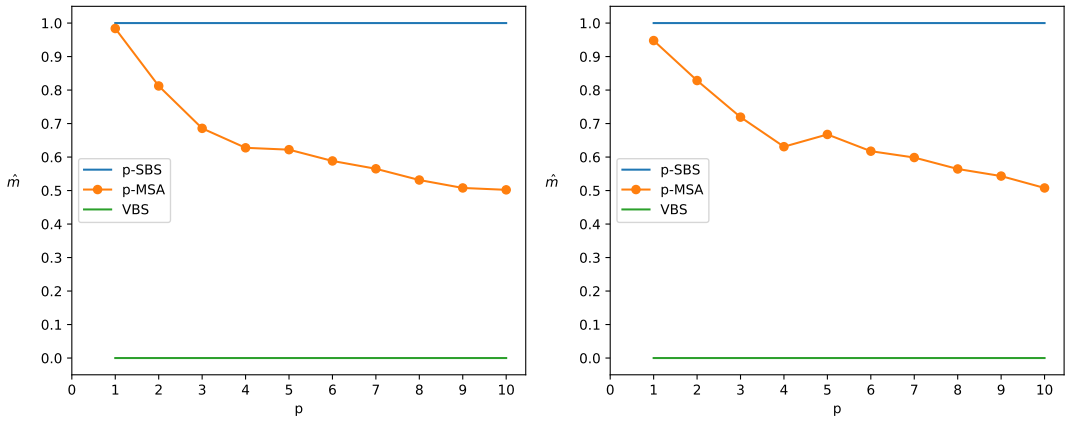
(b) Percentage of wins and average runtime comparison between MSA, CW, and HGS, for L1-norm instances.

TABLE 6 Percentage of best-solved instances per algorithm and average runtime, grouped by benchmark set. Headings legend: % MSA, % CW, % HGS = percentage of instances in the group best solved by MSA, CW, and HGS, respectively. t_MSA, t_CW, t_HGS = average runtime in seconds required to generate a first feasible solution by MSA, CW, and HGS respectively.

Another metric evaluation of the algorithm configurator is a measure that calibrates its performance compared to the SBS and the VSw. This is achieved via the $\hat{m}_s$ metric, as defined in Section 2.7. As proposed in [26], we generalize this evaluation to a version of MSA that incorporates not only one but p values of r for each instance, for p a positive integer. Here we set $p \leq 10$. For that, we trained the configurator to output p best values for r, and build p-MSA, the generalized version of MSA, which computes, p solutions, one for each of the p values predicted by the ML model. The same logic applies for $p-$SBS, a generalized version of SBS, which computes p solutions, using the best p values of r in the training set. The top-ranked p parameter values are found in Table 1 which ranks the parameters in descending order of number of wins on the training set. With this notation, SBS is identical to 1-SBS and MSA is identical to 1-MSA.

The comparison between the solution qualities of $p-$MSA, $p-$SBS, and VBS, expressed as the value of metric $\hat{m}_{p-MSA}$, is given in Figure 6. Here, for both norms, for $p = 1$, we observe that MSA performs only slightly better than the SBS, whereas with increasing values of p the advantage in performance of $p-$MSA as compared to $p-$SBS also increases. This happens all the way until value $p = 10$ for both norms. For larger values of p the performance of

p−MSA and p−SBS are quite even. We notice that the larger the value of p, the closer the quality of the solutions delivered by p−SBS and p−MSA are to the quality of the solutions delivered by the VBS. For $p \geq 4$, the differences in the qualities of the solutions tend to be marginal. This justifies the selection of 4−MSA to be the version to include on our Hierarchical Meta-Solver. Note that even increasing the average time of MSA by a factor of four, it is still much faster than HGS, and takes approximately the same time as CW.



(a) $\hat{m}_s$ values for L2-norm instances. (b) $\hat{m}_s$ values for L1-norm instances.

FIGURE 6 $\hat{m}_s$ values for p-MSA on L2 and L1 instances.

The authors introduced MSA in [10] where both the training set and the testing set were generated from customer addresses in Los Angeles, and in New York city. The distances in those data sets are Euclidean, or L2-norm. The set up here is different in a few ways: Firstly the configurator is part of the selector and both are trained on a training set generated synthetically as discussed in Section 3.2.1. Secondly, the testing set here is unrelated to the training set–it consists of public benchmark sets from the CVRPLib, with distances generated using the L2-norm and the L1-norm. This difference between the sources of the training and testing set results in different performance of the respective ML algorithms: The best-performing ML algorithm in [10] was KNN whereas here it is Gradient Boosting. This difference also affects the $\hat{m}$ performance. The value of $\hat{m}$ here is higher than that in [10] due to the generated training set not sharing the same source as the benchmarks of the testing set. Lastly, in terms of the portfolio, in [10] we compared MSA's performance to that of CW and the first feasible solution of Google's OR-tools' CVRP solver. Here we compare MSA, CW and HGS, since the latter is much superior to Google's OR-tools' CVRP solver.

4.2 | The Effectiveness of the Hierarchical Meta-Solver

The Hierarchical Meta-Solver HMS works as follows. First, it computes the features of the input instance and feeds them to the algorithm selector, that predicts, among 4−MSA, HGS, and CW, which one is expected to work best. We select 4−MSA to include in HMS, since, as discussed in Section 4.1, $p = 4$ offers the best tradeoff between speed and solution quality. When the selected solver is, for example, 4−MSA, the configurator is called and 4−MSA is set to run with the four values of r predicted by the configurator. When the selected solvers are either CW or HGS, then that solver is is run and the output of HMS is the output of the solver in the portfolio it that was selected. The accuracy of the selector is 64.50% for the L2-norm instances and 54.20% for the L1-instances. In general, the mistakes of the

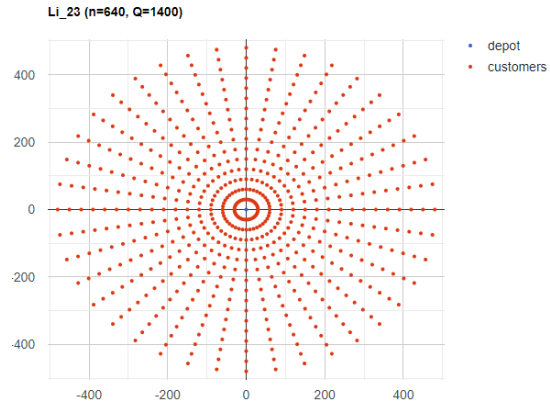


FIGURE 7 Customers and Depot locations for instance Li_23.

selector tend to happen either in instances that are very different from the instances in the training set or when the solution quality of the selected solver is very close to the solution quality of the VBS.

We can trace the errors made by the selector to two sources. One source is in stylized instances that are designed to test extreme cases. Such are the instances in the Li benchmark set. One such example instance is depicted in Figure 7. Because the instances in the training set are of more randomized nature, the selector tends to miss on the selection of the best solver. A second source of errors is when all solvers have similar performance. In this latter case, the errors made by the selector do not affect much the quality of the meta-solver.

In addition to accuracy, we measure the effectiveness of the Hierarchical Meta-Solver HMS by comparing the objective values it delivers to those computed by all the individual solvers in the portfolio. Table 7 shows the percentage of wins and the average values of m_{4-MSA}, m_{CW}, m_{HMS}, and the ones corresponding to the hierarchical meta-solver HMS, m_{HMS}, for each benchmark set. Recall that the m_a metric, defined in Section 2.7, computes the average approximation factor of the objective delivered by a solver a compared to VBS. For CVRP, which is a minimization problem, the closer the value of m_a to 1, the better is the performance.

We consider m_a values that are equal or greater than 5% of the VBS value to indicate poor performance. With this measure of performance, for L2-norm instances, 4−MSA performs poorly for benchmarks B, F, Li, and tai; CW performs poorly across the board with the exception of B and tai; HGS performs poorly on Li and M; and the hierarchical meta-solver performs poorly only on Li. For the L1-norm instances, 4−MSA performs poorly on B, F, Li, and tai; CW performs poorly on almost all benchmarks: A, AGS, B, CMT, E, Golden, M, P, tai, and Uchoa; HGS performs poorly on F, Golden, and Li; and the hierarchical meta-solver performs poorly on only F and Li. This validates the utility of the algorithm selector-configurator that improves greatly on individual solvers in the portfolio.

With the data in Table 7, it is possible to compute the $\hat{m}_s$ values for HMS, which calibrate its performance with the performance of the Virtual Best Solver (VBS) and the Single Best Solver (SBS). For this, we notice that the Single Best Solver, for both norms is 4−MSA, since it has better average m_a values among 4−MSA, HGS, and CW, across all the instances in the testing set. For L2-norm instances $\hat{m}_{HMS} = 0.5$, and for L1-norm instances $\hat{m}_{HMS} = 0.74$. These values indicate that it is harder for HMS to improve the performance of the SBS (4−MSA) in the L1-norm instances than it is to improve the performance of the SBS (4−MSA) in the L2-norm instances. Overall, for both norms, HMS is able to, on average, compute solutions that are around 1% worse than the best computable solutions by any of the solvers in the portfolio.

Bench.	Size	%4-MSA	gap_{4-MSA}	%CW	gap_CW	%HGS	gap_{HGS}	%HMS	gap_{HMS}
A	27	18.52	3.88%	3.70	5.81%	77.78	0.35%	**81.48**	**0.10%**
AGS	10	**90.00**	**0.10%**	0.00	7.45%	10.00	3.41%	**90.00**	**0.10%**
B	23	39.13	6.11%	13.04	4.21%	**47.83**	**3.13%**	**47.83**	**3.13%**
CMT	14	**50.00**	2.02%	0.00	6.45%	**50.00**	1.37%	**50.00**	**1.25%**
E	11	45.45	1.15%	0.00	7.39%	54.55	1.11%	**63.64**	**0.43%**
F	3	0.00	9.63%	0.00	1.33%	**100.00**	**0.00%**	**100.00**	**0.00%**
Golden	20	75.00	0.77%	0.00	10.87%	25.00	3.45%	**80.00**	**0.22%**
Li	12	16.67	4.63%	25.00	9.23%	**58.33**	**4.60%**	16.67	4.77%
M	5	**80.00**	**0.30%**	20.00	5.97%	0.00	5.44%	60.00	1.75%
P	24	**58.33**	**1.13%**	0.00	7.02%	41.67	1.45%	**58.33**	**1.13%**
tai	13	38.46	5.02%	15.38	3.78%	46.15	**1.42%**	**53.85**	**1.32%**
Uchoa	100	57.00	**1.13%**	0.00	7.09%	43.00	2.43%	**69.00**	**1.09%**
All	262	50.38	2.25%	3.82	6.83%	45.80	2.27%	**64.89**	**1.22%**

(a) Percentage of wins and m_a values for 4-MSA, CW, and HGS, and HMS for L2-norm instances.

Benchmark set	#instances	%4-MSA	m_{4-MSA}	%CW	m_{CW}	%HGS	m_{HGS}	%HMS	m_{HMS}
A	27	**55.56**	1.03	3.70	1.07	40.74	1.02	**48.15**	**1.02**
AGS	10	**90.00**	**1.00**	0.00	1.09	10.00	1.04	**90.00**	**1.00**
B	23	39.13	1.06	26.09	1.05	34.78	1.02	**43.48**	**1.02**
CMT	14	**71.43**	**1.01**	0.00	1.07	28.57	1.03	50.00	**1.01**
E	11	**63.64**	**1.01**	0.00	1.07	36.36	1.02	54.55	**1.01**
F	3	33.33	1.08	33.33	**1.01**	33.33	1.05	33.33	1.05
Golden	20	**65.00**	**1.01**	30.00	1.06	5.00	1.13	55.00	1.02
Li	12	16.67	1.06	**66.67**	**1.03**	16.67	1.09	16.67	1.06
M	5	**100.00**	**1.00**	0.00	1.10	0.00	1.03	60.00	1.01
P	24	**70.83**	**1.01**	4.17	1.08	25.00	1.01	45.83	**1.01**
tai	13	23.08	1.05	15.38	1.05	**61.54**	**1.02**	**61.54**	**1.02**
Uchoa	100	63.00	**1.01**	0.00	1.08	37.00	1.03	**75.00**	**1.01**
All instances	262	58.78	1.02	9.54	1.07	31.68	1.04	**59.54**	**1.01**

(b) Percentage of wins and m_a values for 4-MSA, CW, and HGS, and HMS for L1-norm instances.

TABLE 7 Percentage of best-solved instances and accumulated performance metric m_s values per algorithm, grouped by benchmark set. Headings legend: % MSA, % CW, % HGS, % HMS = percentage of best solved instances by MSA, CW, HGS, and HMS respectively. m_{MSA}, m_{CW}, m_{HGS}, m_{HMS} = accumulated performance m_a value for MSA, CW, HGS, and meta-solver HMS respectively.

5 | CONCLUSIONS AND FUTURE WORK

We demonstrate here that using machine learning (ML) for the automatic selection of an algorithm and the automatic configuration of an algorithm's parameters is an approach that can improve significantly the quality of the solution provided. This proof of concept is given here for the capacitated vehicle routing problem (CVRP) where the focus is on selecting very fast algorithms that generate good-quality solutions.

The portfolio of algorithms considered here includes the Hybrid Genetic Search (HGS) algorithm from which the first feasible solution generated is extracted, the Clarke & Right (CW) algorithm as implemented in the state-of-the-art FILO solver [3], and the Meta-Sweep-Algorithm (MSA) which is an automatically configured "Sweep Algorithm".

One novelty in our work is the hierarchical combination of an automatic configurator with an automatic selector. The automatically configured Meta-Sweep-Algorithm (MSA) is fast and highly competitive and its variant 4-MSA, as shown here, dominates the performance of the other algorithms and is the Single Best Solver.

Future research plans include the construction of CVRP meta-solvers with expanded portfolio of algorithms based on local search and exact methods with time limits that are less restrictive than the ones considered here. In that context we will explore *anytime automatic selection* algorithms that predict the best performing algorithm in the portfolio, within a user-specified time limit.

References

[1] M.F. Abdelatti and M.S. Sodhi, *An improved gpu-accelerated heuristic technique applied to the capacitated vehicle routing problem*, Proceedings of the 2020 Genetic and Evolutionary Computation Conference, 2020, pp. 663–671.

[2] L. Accorsi, A. Lodi, and D. Vigo, *Guidelines for the computational testing of machine learning approaches to vehicle routing problems*, Oper. Res. Lett. **50** (2022), 229–234.

[3] L. Accorsi and D. Vigo, *A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems*, Transp. Sci. **55** (2021), 832–856.

[4] F. Alesiani, G. Ermis, and K. Gkiotsalitis, *Constrained clustering for the capacitated vehicle routing problem (cc-cvrp)*, Appl. Artificial Intelligence (2022), 1–25.

[5] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, and A.J. Aljaaf, *A systematic review on supervised and unsupervised machine learning algorithms for data science*, Supervised unsupervised learning data science (2020), 3–21.

[6] E. Alpaydin, *Machine learning*, MIT Press, 2021.

[7] C. Ansótegui, J. Gabas, Y. Malitsky, and M. Sellmann, *Maxsat by improved instance-specific algorithm configuration*, Artificial Intelligence **235** (2016), 26–39.

[8] D.L. Applegate, R.E. Bixby, V. Chvátal, W. Cook, D.G. Espinoza, M. Goycoolea, and K. Helsgaun, *Certification of an optimal tsp tour through 85,900 cities*, Oper. Res. Lett. **37** (2009), 11–15.

[9] F. Arnold, M. Gendreau, and K. Sörensen, *Efficiently solving very large-scale routing problems*, Comput. operations research **107** (2019), 32–42.

[10] R. Asın-Achá, O. Goldschmidt, D.S. Hochbaum, and I.I. Huerta, *Fast algorithms for the capacitated vehicle routing problem using machine learning selection of algorithm's parameters*, Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, 2022, pp. 29–39.

[11] P. Augerat, D. Naddef, J. Belenguer, E. Benavent, A. Corberan, and G. Rinaldi, *Computational results with a branch and cut code for the capacitated vehicle routing problem*, Tech. report, Institut National Polytechnique, 38 - Grenoble (France), 1995.

[12] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H.H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren, *ASlib: A Benchmark Library for Algorithm Selection*, Artificial Intelligence J. (AIJ) (2016), 41–58.

[13] A. Bogyrbayeva, M. Meraliyev, T. Mustakhov, and B. Dauletbayev, *Learning to solve vehicle routing problems: A survey*, arXiv preprint arXiv:2205.02453 (2022).

[14] J. Bossek, P. Kerschke, A. Neumann, M. Wagner, F. Neumann, and H. Trautmann, *Evolving diverse tsp instances by means of novel and creative mutation operators*, Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms, 2019, pp. 58–71.

[15] J. Bossek and F. Neumann, *Exploring the feature space of tsp instances using quality diversity*, Proceedings of the Genetic and Evolutionary Computation Conference, 2022, pp. 186–194.

[16] W. Bożejko, A. Gnatowski, T. Niżyński, M. Affenzeller, and A. Beham, *Local optima networks in solving algorithm selection problem for tsp*, International Conference on Dependability and Complex Systems, 2018, pp. 83–93.

[17] L. Breiman, *Bagging predictors*, Machine learning **24** (1996), 123–140.

[18] L. Breiman, *Random forests*, Machine learning **45** (2001), 5–32.

[19] V. Bulitko, *Evolving real-time heuristic search algorithms*, Artificial Life Conference Proceedings 13, 2016, pp. 108–115.

[20] N. Burkart and M.F. Huber, *A survey on the explainability of supervised machine learning*, J. Artificial Intelligence Res. **70** (2021), 245–317.

[21] R. Carbone and J.S. Armstrong, *Note. evaluation of extrapolative forecasting methods: results of a survey of academicians and practitioners*, J. Forecasting **1** (1982), 215–217.

[22] M. Christofides, *Toth, 1979 christofides n., mingozzi a., toth p., the vehicle routing problem*, Combinatorial Optim., John Wiley Sons, Lond. (1979).

[23] N. Christofides and S. Eilon, *An algorithm for the vehicle-dispatching problem*, J. Oper. Res. Soc. **20** (1969), 309–318.

[24] G. Clarke and J.W. Wright, *Scheduling of vehicles from a central depot to a number of delivery points*, Oper. research **12** (1964), 568–581.

[25] R. Dondo and J. Cerdá, *A sweep-heuristic based formulation for the vehicle routing problem with cross-docking*, Comput. Chemical Eng. **48** (2013), 293–311.

[26] I. Dunning, S. Gupta, and J. Silberholz, *What works best when? a systematic evaluation of heuristics for max-cut and qubo*, INFORMS J. Comput. **30** (2018), 608–624.

[27] R. Elshaer and H. Awad, *A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants*, Comput. Indus. Eng. **140** (2020), 106242.

[28] J. Fellers, J. Quevedo, M. Abdelatti, M. Steinhaus, and M. Sodhi, *Selecting between evolutionary and classical algorithms for the cvrp using machine learning: optimization of vehicle routing problems*, Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2021, pp. 127–128.

[29] M.L. Fisher, *Optimal solution of vehicle routing problems using minimum k-trees*, Oper. research **42** (1994), 626–642.

[30] E. Fix and J.L. Hodges, *Discriminatory analysis: nonparametric discrimination: consistency properties*, Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques (1991), 32–39.

[31] J.H. Friedman, *Greedy function approximation: a gradient boosting machine*, Ann. statistics (2001), 1189–1232.

[32] B.E. Gillett and L.R. Miller, *A heuristic algorithm for the vehicle-dispatch problem*, Oper. research **22** (1974), 340–349.

[33] B.L. Golden, E.A. Wasil, J.P. Kelly, and I.M. Chao, *The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results*, Fleet management logistics (1998), 33–56.

[34] M. Haimovich and A.H. Rinnooy Kan, *Bounds and heuristics for capacitated routing problems*, Math. operations Res. **10** (1985), 527–542.

[35] I.I. Huerta, D.A. Neira, D.A. Ortega, V. Varas, J. Godoy, and R. Asín-Achá, *Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection*, Expert Syst. with Appl. **187** (2022), 115948.

[36] P. Kerschke, H.H. Hoos, F. Neumann, and H. Trautmann, *Automated algorithm selection: Survey and perspectives*, Evolutionary computation **27** (2019), 3–45.

[37] P. Kerschke, L. Kotthoff, J. Bossek, H.H. Hoos, and H. Trautmann, *Leveraging tsp solver complementarity through machine learning*, Evolutionary computation **26** (2018), 597–620.

[38] L. Kotthoff, P. Kerschke, H. Hoos, and H. Trautmann, *Improving the state of the art in inexact tsp solving using per-instance algorithm selection*, International Conference on Learning and Intelligent Optimization, 2015, pp. 202–217.

[39] G. Laporte, M. Gendreau, J.Y. Potvin, and F. Semet, *Classical and modern heuristics for the vehicle routing problem*, Int. transactions operational research **7** (2000), 285–300.

[40] F. Li, B. Golden, and E. Wasil, *Very large-scale vehicle routing: new test problems, algorithms, and results*, Comput. Oper. Res. **32** (2005), 1165–1179.

[41] S. Lin and B.W. Kernighan, *An effective heuristic algorithm for the traveling-salesman problem*, Oper. research **21** (1973), 498–516.

[42] M. Lindauer, H.H. Hoos, F. Hutter, and T. Schaub, *Autofolio: An automatically configured algorithm selector*, J. Artificial Intelligence Res. **53** (2015), 745–778.

[43] M. Lindauer, J.N. van Rijn, and L. Kotthoff, *The algorithm selection competitions 2015 and 2017*, Artificial Intelligence **272** (2019), 86–100.

[44] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, and T. Stützle, *The irace package: Iterated racing for automatic algorithm configuration*, Oper. Res. Perspectives **3** (2016), 43–58.

[45] D. Müller, M.G. Müller, D. Kress, and E. Pesch, *An algorithm selection approach for the flexible job shop scheduling problem: Choosing constraint programming solvers through machine learning*, Eur. J. Oper. Res. **302** (2022), 874–891.

[46] T. Paolo and D. Vigo, *The vehicle routing problem, siam monographs on discrete mathematics and applications*, Soc. Indus. Appl. Math. (2002).

[47] A.A. Pessoa, M. Poss, R. Sadykov, and F. Vanderbeck, *Branch-cut-and-price for the robust capacitated vehicle routing problem with knapsack uncertainty*, Oper. Res. **69** (2021), 739–754.

[48] J.R. Quinlan, *Induction of decision trees*, Machine learning **1** (1986), 81–106.

[49] R.E. Schapire, *Using output codes to boost multiclass learning problems*, ICML, Vol. 97, 1997, pp. 313–321.

[50] R.E. Schapire, "*Explaining adaboost*," *Empirical inference*, Springer, 2013, pp. 37–52.

[51] M. Seiler, J. Pohl, J. Bossek, P. Kerschke, and H. Trautmann, *Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem*, International Conference on Parallel Problem Solving from Nature, 2020, pp. 48–64.

[52] D. Sigurdson and V. Bulitko, *Deep learning for real-time heuristic search algorithm selection*, Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2017.

[53] M. Steinhaus, *The application of the self organizing map to the vehicle routing problem*, University of Rhode Island, 2015.

[54] M. Stone, *Cross-validatory choice and assessment of statistical predictions*, J. royal statistical society: Ser. B (Methodological) **36** (1974), 111–133.

[55] S. Strassl and N. Musliu, *Instance space analysis and algorithm selection for the job shop scheduling problem*, Comput. Oper. Res. **141** (2022), 105661.

[56] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, *New benchmark instances for the capacitated vehicle routing problem*, Eur. J. Oper. Res. **257** (2017), 845–858.

[57] T. Vidal, *Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood*, Comput. Oper. Res. **140** (2022), 105643.

[58] T. Vidal, T.G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, *A hybrid genetic algorithm for multidepot and periodic vehicle routing problems*, Oper. Res. **60** (2012), 611–624.

[59] K. Zhao, S. Liu, J.X. Yu, and Y. Rong, *Towards feature-free tsp solver selection: A deep learning approach*, 2021 International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1–8.

[60] X. Zhu and A.B. Goldberg, *Introduction to semi-supervised learning*, Synthesis lectures artificial intelligence machine learning **3** (2009), 1–130.