

## ENSEMBLE MACHINE LEARNING-BASED STOPPING RULE FOR GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

GUILHERME CAEIRO DE MATTOS<sup>1,\*</sup>, LEOPOLDO ANDRÉ DUTRA LUSQUINO FILHO<sup>2,3</sup>,  
LUIDI GELABERT SIMONETTI<sup>1</sup> AND PRISCILA MACHADO VIEIRA LIMA<sup>1,4</sup>

**Abstract.** As with many metaheuristics, the Greedy Randomized Adaptive Search Procedure (GRASP) lacks an effective stopping rule in its standard form and relies on ineffective criteria. This often leads to a waste of computational resources. To address this limitation, rules based on Bayesian statistics, cumulative distribution function, extreme value theory, and machine learning algorithms have been proposed in the literature. However, these methods also present shortcomings, as they may fail on certain instance types or be computationally expensive. In response, this work seeks to better understand these shortcomings and overcome some of them through an ensemble-based machine learning approach. To demonstrate its capabilities, the new rule was evaluated on a custom dataset composed of execution data from three optimization problems and compared to a group of alternatives. The evaluation used cross-validation and an additional test designed to assess generalization across problems, in which the model was trained on two optimization problems and tested on a third. Two custom metrics focused on evaluating how well the rules stop the metaheuristic at predetermined points in the search are also introduced. The results indicate that the proposed stopping rule is competitive on harder instances.

**Mathematics Subject Classification.** 90C59, 68T01.

Received June 21, 2025. Accepted January 14, 2026.

### 1. INTRODUCTION

#### 1.1. Background

Metaheuristics are search algorithms that employ a set of procedures to form a generic search strategy to explore the solution space of optimization problems. However, despite being capable of achieving quality solutions for computationally difficult problems, they are commonly unable to measure that quality. For that reason, many lack an effective stopping rule capable of leveraging that information and rely on simple criteria, like a maximum number of iterations, a number of iterations without improvement in the best solution found,

---

*Keywords.* Stopping rule, GRASP, metaheuristics, machine learning.

<sup>1</sup> PESC/COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil.

<sup>2</sup> Institute of Science and Technology, São Paulo State University, Sorocaba, SP, Brazil.

<sup>3</sup> Recod.ai Lab, Institute of Computing, State University of Campinas, Campinas, SP, Brazil.

<sup>4</sup> Tercio Pacitti Institute (NCE), Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil.

\*Corresponding author: [mattosgc@cos.ufrj.br](mailto:mattosgc@cos.ufrj.br)

or similar conditions associated with time. As a consequence, the algorithm may waste time and computational resources by continuing the search for an improvement for longer than needed or by stopping while there is still a high chance of finding a better solution in the next iteration, as pointed out by Ribeiro *et al.* [57] and Ghoreishi *et al.* [23].

In the case of the standard GRASP metaheuristic, it suffers from that problem [57], but the literature already provides multiple stopping rules for it that aim to be effective without requiring major modifications to the way the metaheuristic works. Those alternatives are based on Bayesian statistics [44, 45, 48], cumulative distribution function [19, 57], extreme value theory [13], and machine learning [38]. Although these approaches allow the metaheuristic to be stopped based on an informed decision about the state of the optimization process, they also have their own limitations.

## 1.2. Literature gap

It is possible to identify multiple limitations in the existing stopping rules for GRASP, some of which are presented by their own authors, and others that are identified in the present work. For example, in the case of rules that employ as a criterion the probability of improvement estimated by a cumulative distribution function (CDF), they may perform well [19] if a parametric family of distributions can be properly fitted to the distribution of solution cost values; otherwise, they are likely to fail. Additionally, limitations may also be related to the cost of applying a given stopping rule. That situation can be exemplified by the method of Mattos [38], which replaces the CDF with a machine learning model in the task of estimating the probability of improvement in future iterations. In this case, the problem lies in the fact that it requires a costly process to obtain the training data. Another problem with that stopping rule is that the model is trained on data from a single optimization instance, which leads to significant variation in performance, depending on which instance is used for training and which are used for evaluation. However, its authors briefly mention the possibility of combining models in an ensemble to leverage their performance across different subsets of instances.

Given that situation, the limitations of existing stopping rules create an opportunity for the development of a new alternative that overcomes the shortcomings of current methods. One possible way to do that is to use one of the existing rules as basis and improve it. Among the options available, one that might be among the most promising is the method based on machine learning algorithms, due to the flexibility that it can achieve depending on the training data.

With that said, the use of machine learning as a component of a stopping rule is an example of the use of machine learning to support a metaheuristic, a practice that is not limited to GRASP. That combination can be seen in other works, such as in Bożejko *et al.* [9], where a neural network replaces the tabu list in the metaheuristic Tabu Search, or in Alicastro *et al.* [2], where Q-Learning is used to select the best neighborhood structure in the local search phase of the metaheuristic Iterated Local Search. Alternatively, the use of metaheuristics in support of machine learning is also common, with them being applied, for example, in hyperparameter optimization (HPO) [1, 42, 66] and feature selection [67]. These techniques partially automate the training process and could be applied to any machine learning approach to a stopping rule for GRASP.

## 1.3. Contributions

Motivated by the context provided in Sections 1.1 and 1.2, the present work sought to understand the limitations of some of the existing effective stopping rules for GRASP and to leverage ensemble learning to propose an ensemble-based alternative that improves on some of those shortcomings or on the general performance of those effective methods. Given these objectives, this paper contributes to the literature surrounding stopping rules for GRASP in three ways. Firstly, it provides a discussion about the limitations of some existing methods, covering those already mentioned by previous works, and demonstrating others that were not discussed previously. Building on this analysis, the second contribution consists of the proposal of an alternative stopping rule based on the method of Mattos [38], where the estimators based on single instances are replaced by an ensemble model that is aware of instance difficulty. That approach is taken in an attempt to improve the generalization

capability of the original method and also to outperform existing alternatives. The training utilized hyperparameter optimization, and the proposed method had its performance compared to a set of past stopping rules for GRASP based on CDF and on machine learning. The results indicate that the proposed method was better or comparable to existing methods in certain cases, especially in harder instances.

The third and final contribution consists of two custom metrics, named Similarity at Threshold (SAT) and Mean Similarity at Threshold (MSAT). They were included to measure how well the estimators perform at stopping the metaheuristic execution at given levels of solution quality.

## 1.4. Text outline

In the remainder of this text, in Section 2, the existing stopping rules for GRASP are presented, as well as other pertinent related works. Next, in Section 3, the ensemble-based stopping rule is introduced, and in Section 4, the evaluation approaches and their results are described. This paper is concluded in Section 5, where possible future work is also mentioned. Finally, a discussion on the limitations of some existing stopping rules, which explores the research gap attacked by this text, is provided in Appendix A.

## 2. RELATED WORK

### 2.1. GRASP

The *Greedy Randomized Adaptive Search Procedure* (GRASP) [20] is a simple metaheuristic that is based on two phases that are performed at each iteration. The first is the construction phase, where an initial solution is built through a greedy randomized search, and the second is the local search phase, where a local search procedure is used to look for an improvement in the solution space in the neighborhood of the initial solution. Those two phases are illustrated in Algorithm 1.

In the constructive phase, the greedy randomized search is close to a usual greedy search, where, at each vertex explored in the search graph (in this case, a search trying to create an initial feasible solution), the next vertex to be visited is the one that brings the biggest improvement towards the objective function among the available options. However, that procedure is randomized because, instead of always picking the best next vertex, it randomly picks one in the list of best options. In the context of GRASP, that list is called *Restricted Candidate List* (RCL), and its size is determined by a parameter called *RCL Size*. The RCL size can be a number specified by the user or a percentage. In the latter case, Resende and Ribeiro [55] describes a quality-based selection where, in a minimization setting, a vertex with cost  $c$  is chosen to compose the RCL if it satisfies the condition  $c_{\min} \leq c \leq c_{\min} + \alpha(c_{\max} - c_{\min})$ . Here,  $c_{\min}$  and  $c_{\max}$  are the minimum and the maximum costs among the options, and  $\alpha$  is a value between 0 and 1 that represents the RCL size. That parameter affects the exploration and exploitation capabilities of the metaheuristic, where a high size leads to a more explorative behavior and a small size leads to more exploitation.

After the constructive phase is performed and an initial solution is produced, the local search investigates the neighborhood of that solution. That kind of procedure usually uses one of two strategies: first improvement, where it stops after finding a solution better than the initial, or best improvement, which only stops after investigating the entirety of the neighborhood (Resende and Ribeiro [55] provides examples of local search strategies for a number of optimization problems). Once this phase is finished, the next step is to evaluate the stopping condition, which terminates the execution if it is met; otherwise, the metaheuristic proceeds to the next iteration. Due to the simplicity of the two phases of the metaheuristic, it can require only two parameters in its standard form, the RCL size and the stopping criterion, as shown in Algorithm 1.

The standard GRASP can also be improved in some ways. One of them is through the use of path-relinking [56]. In that method, a set with elite solutions (the  $e$  sufficiently different best solutions found) is kept, and at each iteration, the path between its members and the iteration's solution is followed (in one or both ways) by producing solutions through gradual changes in the direction of the target solution. Alternatively, the path-relinking can be executed just as a post-optimization process, exploring the path between the solutions in the

elite set after the GRASP execution ends. Other modifications that are worth mentioning in the present work are the Reactive GRASP [53], which modifies the constructive phase so that the RCL size is selected dynamically, allowing the metaheuristic to balance exploration and exploitation by itself, and the Lagrangian GRASP [51,52], which is a hybridization of GRASP and Lagrangian Relaxation.

---

**Algorithm 1:** Pseudocode for the standard GRASP algorithm.

---

```

procedure grasp (numIters, rclSize)
1  bestSol  $\leftarrow$  null
2  bestCost  $\leftarrow$   $+\infty$ 
3  for i  $\leftarrow$  1 to numIters do
4      sol  $\leftarrow$  greedyRandomizedSearch(rclSize)
5      sol  $\leftarrow$  localSearch(sol)
6      cost  $\leftarrow$  calculateCost(sol)
7      if cost < bestCost then
8          bestCost  $\leftarrow$  cost
9          bestSol  $\leftarrow$  sol
        end
    end
10 return bestSol, bestCost
end

```

---

## 2.2. Stopping rules for GRASP

### 2.2.1. Overview

In this work, an effective stopping rule is one that makes decisions based on information about the optimization process being conducted, such as assessment of solution quality or probability of improvement, while ineffective rules are those that rely solely on predefined conditions that do not consider the state of the optimization process, like a maximum number of iterations or time. In Ghoreishi *et al.* [23], a series of stopping criteria for evolutionary algorithms are presented and the ineffective are grouped into a category that its authors call *direct termination criteria*. Meanwhile, the effective criteria are divided into a series of classes based on what type of information is used. Going beyond evolutionary algorithms, Corominas [16] discusses stopping rules for metaheuristics in general and places some of the ineffective in the category of *budget* rules, because the final solution is expected to be the best that the metaheuristic can find with the resource budget that is allocated at the beginning of the execution. In addition, that work also presents simple formulas that use information about the type of the stopping rule, the optimization problem, and the type of the metaheuristic to determine the maximum number of iterations to be executed. Automated selection of the number of iterations is also utilized in Goli *et al.* [27], which applies a method of experimental design to select metaheuristic parameters.

Given that context, the problem of the lack of an effective stopping rule for the standard GRASP, introduced in Section 1, lies in the fact that it is not possible for the metaheuristic to assert anything about the optimality of a given solution or about any other kind of measure of quality. Consequently, if an ineffective stopping rule is used, computational resources and time are likely to be wasted. In order to illustrate that, two extreme hypothetical examples can be provided. In the first, the optimal solution is found in the first iteration, but the metaheuristic continues the search until the resource budget is exhausted, which means that most of those resources were used unnecessarily. In the second, the search is stopped at the iteration immediately before the one where the optimal solution – or a relevant improvement – would be found, wasting all the computational effort that was made after the most recent change to the incumbent solution (best solution found). Although these extreme situations are not the most likely to occur, intermediate situations, where some amount of resource waste still occurs, are expected.

In an effort to attenuate that limitation (*i.e.*, reduce waste), effective stopping rules were already proposed. Orsenigo and Vercellis [48] introduces an approach based on Bayesian statistics, where two prior distributions were tested for GRASP implementations for the problem of building classification trees. They compare their method to the use of a maximum number of iterations. Their stopping rule allowed the metaheuristic to run long enough to produce a tree capable of reaching accuracy values comparable to those reached by the longest-running GRASP that used a maximum number of iterations. Given that their approach took less time to reach those results, they argue that their method was able to reach an optimal tradeoff between accuracy and computational effort. Another Bayesian stopping rule is also proposed by Neves *et al.* [45] and Neves [44], which implements the method of Boender and RinnooyKan [8] and shows that, for the three stopping criteria covered, all were able to deliver quality solutions and save time in instances of the five optimization problems explored in that work.

In Carling and Han [13], a method based on statistical bounds for the optimal solution is described. In that approach, two point estimators from the Extreme Value Theory are evaluated, and the stopping rule consists of using the lower and upper bounds that they estimate to calculate the optimality gap (the difference between the bounds, divided by the lower bound). That gap can then be compared to a target gap to determine whether to stop the execution or not. In that work, the point estimators are the focus, and no evaluation of the stopping rule is provided. However, it is shown that, depending on the instance under evaluation (the work employs a group of instances from four optimization problems), their testing parameters were not enough to guarantee that the bounds cover the optimal solution. The authors also present their work as complementary to the probabilistic stopping rule of Ribeiro *et al.* [57].

The probabilistic stopping rules for GRASP [19, 57] consist of approaches that make an assumption about the distribution of the cost value of iteration solutions. Based on that, the cumulative distribution function (CDF) of that distribution is used to estimate the probability of finding a solution at least as good as the current minimum in subsequent iterations. These stopping rules stop the execution if that probability reaches a target threshold  $\beta$ . Given some limitations of Ribeiro *et al.* [57] (discussed in Appendix A), Mattos *et al.* [40] and Mattos [38] propose a machine learning-based approach that replaces the CDF with tree-based machine learning models. They show that those estimators can achieve better results in multiple instances of two different optimization problems.

Besides the stopping rules presented in this section, no recent alternatives have been proposed for GRASP. However, Neves *et al.* [45] and Neves [44] also mention some that are not focused on that metaheuristic, but that might be applicable. Additionally, it is also worth mentioning that the Lagrangian GRASP, briefly mentioned in Section 2.1, already provides lower and upper bounds for the optimal solution, and, for that reason, a stopping rule based on the optimality gap can be used [55]. Although unrelated to stopping rules for GRASP, Arbelaez and O’Sullivan [3] propose a machine learning-based stopping rule for local search procedures. This approach is indirectly relevant because the local search phase is a core component of the GRASP metaheuristic. The rule they propose focuses on the Cable Routing Problem (CRP) and the Traveling Salesman Problem (TSP).

In the remainder of this section, some stopping rules for GRASP are covered in greater detail. Sections 2.2.2 and 2.2.3 present the probabilistic stopping rules based on the normal and on the gamma distributions, respectively, while Section 2.2.4 focuses on the approach based on machine learning. Those methods also received additional attention in Appendix A, which provides a discussion on the limitations of some of the approaches mentioned in the present section. The Bayesian stopping rules, however, are not covered in the discussion because no public implementation of them was found and no recent work reported the use of any of them.

### 2.2.2. Probabilistic stopping rule based on normal distribution

In Ribeiro *et al.* [57], the authors observed that, for a group of instances of the Quadratic Assignment Problem (QAP), the Set  $k$ -Cover Problem (SCP), the Uncapacitated  $p$ -Median Problem (PMP), and the 2-path Network Design Problem (2PNDP), the distributions of their cost values appeared to be approximately normal. For that reason, given the mean  $\mu$  and the standard deviation  $\sigma$  of the costs observed up to the iteration  $k$  being evaluated, as well as the minimum cost  $c_{\min}$  observed in the same interval, they proposed the use of the CDF

of the normal distribution to estimate the probability of finding a solution smaller than or equal to  $c_{\min}$ . That CDF is given by equation (1), where  $f(c, \mu, \sigma)$  is the probability density function (PDF).

$$F(c_{\min}, \mu, \sigma) = \int_{-\infty}^{c_{\min}} f(c, \mu, \sigma) dc. \quad (1)$$

After the value of the CDF is obtained, the stopping method consists of comparing it to a threshold  $\beta$ , which is a target probability provided by the user. If  $F(c_{\min}, \mu, \sigma) \leq \beta$ , the execution is stopped; otherwise it continues. In order to save computational resources, it is also recommended that the calculation of CDF should be performed only at iterations where there is a change in the minimum or periodically.

In addition to that approach, the proponents of the method also suggested the use of the truncated normal distribution to estimate a more precise probability. By doing so, they managed to achieve good results for the PMP and the 2PNDP instances in tests using the costs of the best known solutions from the literature (optimal or not) as lower bounds. However, for the QAP and the SCP instances, they were unable to reach some of the thresholds used for testing.

The same work also mentions the possibility of a stopping rule based on the number of iterations needed to find a new solution that improves the current minimum by a given amount. An example supporting this possibility was also provided, but that approach was left as a subject for future research.

### 2.2.3. Probabilistic stopping rule based on gamma distribution

In Felici *et al.* [19], it was observed that, for the Minimum Cost Satisfiability Problem (MinCostSAT), the distribution of the cost values showed a significant negative skewness, motivating its authors to propose a stopping rule inspired by Ribeiro *et al.* [57] (described in Sect. 2.2.2), but based on two phases. The first, which they called *fitting-data*, consists in identifying a parametric family of probability distributions that is suited to represent the empirical distribution of cost value. Using Maximum Likelihood Estimation (MLE), the method estimates the best parameters to fit the chosen distribution to the data. The second phase, called *improve-probability*, uses the CDF to estimate the probability of improvement. It decides whether to stop the execution or not by comparing the probability to a threshold provided by the user.

In the empirical study conducted in that work, the authors argued that, in minimization problems, it is easier to find solutions with higher costs and difficult to reach those with smaller costs. For that reason, it is expected to find a higher number of solutions with costs between the mean and the maximum costs (which means a negatively skewed distribution). In order to deal with this situation, in the *fitting-data* phase, they proposed the use of a procedure that reflects the distribution, which is presented in equation (2). In that equation,  $C$  is the set of all costs observed, and  $\bar{c}$  are the reflected costs.

$$\bar{c} = \max(C) - c, \quad \forall c \in C \quad (2)$$

Next, for their MinCostSAT instances, they observed that the reflected distributions resembled the gamma distribution and proposed the use of that parametric family. Given the reflection procedure, they proposed the calculation of a modified CDF that accounts for it in the *improve-probability* phase. In equation (3), that CDF is presented, where  $\bar{c}_{\max}$  is the maximum value among the reflected costs (the original minimum),  $f(\bar{c}_{\max}, a, \theta)$  is the PDF of the gamma distribution, and  $a$  and  $\theta$  are the shape and scale parameters.

$$F'(\bar{c}_{\max}, a, \theta) = 1 - \int_0^{\bar{c}_{\max}} f(c, a, \theta) dc. \quad (3)$$

In their implementation of the proposed method, the authors estimated the parameters of the distribution only once (using MLE), based on an initial sample of cost values whose size was defined by the user. After that, the CDF was always calculated based on those parameters and only in iterations where minimum changes occurred. According to the results obtained for that method, the stopping rule was able to achieve a significant

reduction in execution time in the vast majority of the test instances, with a minor negative impact on the quality of the solutions.

The use of the method in the literature is limited. In Alicastro *et al.* [2], it was applied to a variation of Iterated Local Search called Reinforcement Learning Iterated Local Search, while Pastore *et al.* [49] and Ferone *et al.* [21] only considered it as a possible alternative to the stopping rule that was used.

#### 2.2.4. Machine learning-based stopping rule

The machine learning-based stopping rule for GRASP [38] is a method that replaces the CDF of the normal distribution in the approach presented in Section 2.2.2 with a machine learning model that estimates the probability of improvement or the number of expected future minima. In that method, the model is trained on a dataset composed of iteration data obtained from a number of GRASP executions. Those executions result from using a GRASP implementation to solve a given instance of an optimization problem. After training, the model is then used to evaluate GRASP iterations from unknown instances of the same problem or belonging to other problems, and stops the execution if the estimated value reaches the threshold provided by the user. That method has similarities with the stopping rule for local searches proposed by Arbelaez and O’Sullivan [3]. However, it aims to be problem independent (which varies depending on the feature combination employed) and uses as its target the two previously mentioned alternatives rather than an expected cost for the final solution.

Regarding those targets, the way they are calculated starts with counting the number of improved solutions in the interval between the current iteration and the last iteration of the execution. In other words, for an execution of size  $10^6$  (size used by Mattos [38]) and an iteration  $k$ , the interval is  $[k + 1, 10^6]$ . If the desired target is a probability of improvement, the number of improved solutions found in that interval is divided by the size of the execution. On the other hand, if the expected target is the number of remaining minima, the division is simply avoided.

As for the arguments for the fixed upper bound of the counting interval, the first consists of not having to run each execution for more than  $10^6$  iterations, which saves time during data acquisition. The second argument is the observation that executions usually do not show many minimum changes after iteration  $10^5$ , meaning that, in a sliding interval of fixed size, the number of remaining minima observed after that iteration would not change by much. However, the same work also proposed that, in a future work, it would be pertinent to evaluate the impact of the tested approach in comparison to one that always considers a counting window of size  $10^6$ .

In the tests performed, it was reported that the used method achieved satisfactory performance from the perspective of the metric Root Mean Squared Error in some instances. It happened in those belonging to the same optimization problem used for training, as well as in some belonging to different problems, depending on the training features employed. When compared to the CDF of the truncated normal distribution, it presented a smaller error on multiple occasions (especially in SCP instances; it was also tested on QAP instances). It was also briefly mentioned that an ensemble could be used to combine models trained on different feature sets, with the aim of leveraging their complementary performance across different subsets of the data.

Lastly, it is important to note that Mattos [38] left multiple improvement paths open. In addition to the previously mentioned use of a fixed counting window, it also suggests the tightening of the normalization boundaries; finding the ideal dataset size, with the aim of possibly reducing the amount of data required for training; checking the feasibility of online learning; verifying the applicability of the method to other optimization problems; and testing additional training features.

### 2.3. Integration of machine learning and metaheuristics

As mentioned in Section 1, machine learning can be used to support metaheuristics. Those combined techniques are known as Learning-based Intelligent Optimization Algorithms (LIOA) [34], where *Intelligent Optimization Algorithms* (IOA) commonly refers to nature-inspired metaheuristics. However, IOA has a broader definition and, according to Tao *et al.* [62], also includes neighborhood search algorithms, a category where GRASP can be included. That work also defines that IOAs can be classified into two other categories: evolutionary and based on swarm intelligence. Moving from IOAs to LIOAs, some examples can be provided.

Starting with evolutionary methods, in Goli [25], the Hybrid Red Deer and Genetic Algorithm (HRDGA) is proposed, and  $K$ -Means [36] is applied to help classify the individuals as male or female red deer in the metaheuristic's Red Deer Algorithm [18] segment. Sulaman *et al.* [60], on the other hand, tries to reduce the fitness calculation time for a Differential Evolution algorithm by using a Random Forest Regressor as a surrogate for the fitness function, in a paradigm known as Surrogate-assisted Evolutionary Algorithms (SAEA). That paradigm is applied to situations where the evaluation of an objective function is expensive, such as in Simulation-based Optimization [35], because simulations are time-consuming and complicated [28].

When it comes to swarm intelligence LIOAs, Sun *et al.* [61] propose an enhancement to the Ant Colony Optimization (ACO) algorithm for the Orienteering Problem (a variant of the TSP) that they call ML-ACO. In that method, a machine learning model is used to estimate the probability that a given edge in the instance graph belongs to the optimal solution and uses that information to warm start the ACO. A series of examples of swarm-based LIOAs is also provided by Karaboga *et al.* [33], which surveys techniques that apply machine learning to enhance the Artificial Bee Colony (ABC) algorithm.

As for neighborhood search LIOAs, the methods of Bożejko *et al.* [9] and Alicastro *et al.* [2] mentioned in this text's introduction fit that category, as well as a GRASP implementation coupled with the machine learning-based stopping rule of Mattos [38]. The models proposed in Mattos [38] are each trained on data from a single optimization instance, making them highly specialized. This raises concerns related to generalization, which are covered in Appendix A and further demonstrated in Section 4.4. However, there are ways to combine such specialist models using ensemble learning.

Carvalho *et al.* [15] review works on Dynamic Regressor Selection (DRS), an ensemble method where models are chosen adaptively. One such work is Moura *et al.* [43], which classifies DRS approaches into three categories: (1) *dynamic selection*, where only the most appropriate regressor is chosen to provide the output; (2) *dynamic weighting*, where predictions from all regressors are combined in a weighted manner; and (3) *dynamic weighting with selection*, where a subset of the regressors is selected and have their outputs combined in a weighted manner. That work notes that each regressor is specialized in one region of competence (an area of the feature space). As such, the model that is chosen to produce the output of the ensemble is the one whose region of competence encompasses the observation being evaluated. In Cabral *et al.* [12], the dynamic selection is performed through the use of classifiers that learn how to map features to a regressor. Among the classification algorithms that it tests, Random Forest [10] (classifier) and  $K$ -Nearest Neighbors [63] are pertinent to the present work.

Given an ensemble architecture capable of combining specialist regressors, a challenge that remains is how to choose the hyperparameters and the features used by the models that compose the ensemble. Choosing appropriate hyperparameter values can have a significant impact on prediction quality [66], and the same applies to feature selection. In the case of hyperparameter optimization (HPO), one of the most commonly used methods is Grid Search [1], where all possible value combinations are tested (possibly restricted to a subset of the hyperparameters and with constraints to valid value ranges). However, that exhaustive search might require a prohibitive amount of time, and, for that reason, more efficient approaches are desirable. In Wicaksono and Afif [66] and Alibrahim and Ludwig [1], the approach taken is the use of metaheuristics, specifically Genetic Algorithm [31] (GA). Their results show that this choice resulted in a significant reduction in search time in comparison to Grid Search, without compromising solution quality.

In feature optimization, finding an optimal combination of features can be done through multiple methods, including metaheuristics. In Yusta [67], among other algorithms, Genetic Algorithm and GRASP are tested, with the latter achieving the best results in most of the test cases.

According to Morales-Hernández *et al.* [42], HPO can be single-objective or multi-objective. In single-objective, there is a single objective function that estimates the performance of the model, while in multi-objective there are multiple functions, which calculate multiple performance measures. The same work states that, in cases of multi-objective optimization, the objectives are often scalarized into a single one. It also mentions that the number of features can serve as a complexity measure in the context of balancing training cost and error-based performance.

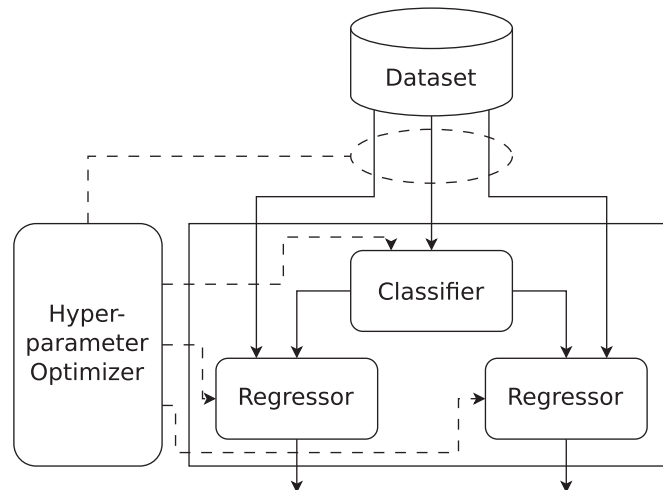


FIGURE 1. Proposed ensemble architecture. Arrows indicate the data flow of the ensemble, while dashed arrows indicate where the HPO acts.

Regarding scalarization, it can be done in multiple ways [25, 26, 46, 68]. Among those methods, a popular choice [26] is the weighted sum, which is defined by  $\max \sum_{i=1}^m w_i f_i(x)$ , subject to  $x \in X$ , where  $X$  is the set of feasible input combinations,  $m$  is the number of objective functions,  $f_i(x)$  is the  $i$ -th objective function, and  $w_i$  is the weight associated with it. In addition to that, Žižović *et al.* [68] also covers two multiplicative methods: the weighted product  $\max \prod_{i=1}^m w_i f_i(x)$  and the product of exponents  $\max \prod_{i=1}^m f_i(x)^{w_i}$ . In those three methods, it is assumed that  $\sum_{i=1}^m w_i = 1$ . For the sake of the present work, the equations are presented as maximization. In Goli [24], the weighted sum is also used to combine an objective to be minimized and an objective to be maximized.

Considering that HPO involves training multiple models in order to find a good – potentially optimal – hyperparameter configuration, it is pertinent to take advantage of the parallel-processing capabilities of modern computers. In the case of GA, Luque and Alba [37] show that parallelism can be achieved through a multi-island architecture, where multiple populations are evolved separately in islands running on separate computers or processor cores. Periodically, a migration procedure occurs between islands, where one or more solutions from each island can move to other islands. According to the same work, the four parameters of a migration policy are the migration gap, which is the number of generations between migrations; the migration rate, which is the number of solutions to be moved; the migrant selection method; and the topology of the algorithm, which stipulates the connections between islands.

### 3. METHODOLOGY

#### 3.1. Model architecture

Given the limitations presented in Appendix A (primarily those of the original machine learning-based stopping rule), this work proposes an ensemble-based alternative inspired by the approach of Mattos [38], which was described in Section 2.2.4. In this alternative, the model also replaces the CDF as the probability or future minima estimator but employs a structure that takes into consideration instance difficulty (as defined in Appendix A) in order to provide more accurate results. The structure is that of an ensemble composed of three models: one classifier at the top and two regressors at its base, as illustrated in Figure 1.

The model at the top is the entry point of the ensemble and is responsible for classifying each iteration as belonging to an easy or a difficult instance. Then, depending on the difficulty estimated, the iteration is

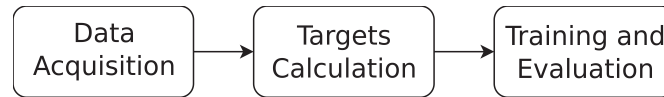


FIGURE 2. Steps of the model creation phase.

forwarded to one of the models at the base. Each of those models is trained on instances of only one difficulty, and, as such, iterations classified as coming from easy instances are redirected to the model trained on easy instances, and iterations classified as coming from difficult instances are evaluated by the model trained on difficult instances. The final output of the ensemble is finally produced by the selected model, and, as in Mattos [38], it can be a probability or the number of future minima (further details are provided in Sect. 4.1).

That kind of model combination represents a form of dynamic regressor selection, because the prediction task is dynamically attributed to a model that is deemed specialized on the type of the observation being evaluated. In this work's approach, the classifier does not use the output of the specialist models directly in its training. The dataset already contains the class of each observation. This sets that approach apart from the ensemble methods in which the identification of the specialists occurs during the training. However, the training of the models is not entirely independent, because it includes the use of a hyperparameter optimizer, which can be employed to select, in combination, the usual hyperparameters of the machine learning algorithms, the training instances, and the training features used by each model. In other words, there is only one objective function for the entire ensemble, which means that the hyperparameters selected for a given classifier or regressor may impact those selected for the other models.

## 3.2. Stopping rule

### 3.2.1. General idea

With the architecture in place, the general process for the stopping rule has two phases: model creation and stopping rule usage. In model creation, three steps are executed, as shown in Figure 2. It starts with the collection of data from different optimization processes related to different types of instances. In this case, *data* comprise sequences of iteration costs and other information obtained by running GRASP to solve a group of instances. *Type* refers to difficulty, cost distribution shapes, and other characteristics of these instances. All this information is stored to be used directly as features or further processed into derived features.

After this initial dataset is obtained, the next step is to assign the instance difficulty to each iteration and calculate the probability of improvement or the number of expected future minima. These values will serve as targets for the models in the ensemble, allowing the execution of the last step, where the ensemble is trained and evaluated. Details about features and targets are available in Section 4.1.

Once the ensemble is deemed ready for practical use, the usage phase starts. In that phase, similarly to what is done in the original probabilistic stopping rule, the ensemble is used to evaluate iterations of executions of unknown instances, and its estimates are compared to a threshold  $\beta$ . For any given iteration, the execution is stopped if the estimate is smaller than or equal to that threshold. Unless needed, the first phase is expected to be executed only once, while the usage phase can be applied to multiple instances. However, over time, situations of reduced generalization or the availability of additional training data might motivate the retraining of the ensemble.

### 3.2.2. Modified GRASP algorithm

In Algorithm 2, a pseudocode covering phase two of the stopping rule process is provided and includes some modifications to the standard GRASP from Algorithm 1. The variables *costsSeq* and *feat* are, respectively, an array containing the costs obtained at each iteration and a data structure containing the features of the previous or of the current iteration. The storage of information about previous iterations is important because

the function *calcFeat*, which calculates the features for the current iteration, may require it depending on the features being used by the ensemble.

After the updated features are obtained, the function *calcRemainMin* uses the ensemble to estimate the number of remaining minima. The execution is then stopped if that estimate is less than or equal to a threshold provided by the user, or if a maximum number of iterations is reached. That additional stopping criterion is optional and is used as a backup in case the threshold is never reached or takes too long for it to happen. A graphical representation of a GRASP loop that incorporates the stopping rule is provided in Figure 3.

---

**Algorithm 2:** GRASP with the proposed stopping rule.

---

```

procedure grasp (maxIters, rclSize,  $\beta$ )
1  | bestSol  $\leftarrow$  null
2  | bestCost  $\leftarrow$   $+\infty$ 
3  | costsSeq  $\leftarrow$   $\{\emptyset\}$ 
4  | feat  $\leftarrow$  null
5  | for  $i \leftarrow 1$  to maxIters do
6  |   | sol  $\leftarrow$  greedyRandomizedSearch(rclSize)
7  |   | sol  $\leftarrow$  localSearch(sol)
8  |   | cost  $\leftarrow$  calculateCost(sol)
9  |   | costsSeq  $\leftarrow$  costsSeq  $\cup$   $\{cost\}$ 
10 |   | if  $cost < bestCost$  then
11 |   |   | bestCost  $\leftarrow$  cost
12 |   |   | bestSol  $\leftarrow$  sol
13 |   | end
14 |   | feat  $\leftarrow$  calcFeat( $i, bestCost, costsSeq, feat$ )
15 |   | remainMin  $\leftarrow$  calcRemainMin(feat)
16 |   | if  $remainMin \leq \beta$  then
17 |   |   | return bestSol, bestCost
18 |   | end
19 |   | end
20 | end
21 | return bestSol, bestCost

```

---

## 4. EXPERIMENTAL EVALUATION

### 4.1. Dataset

#### 4.1.1. Dataset generation

The present work employed two datasets in the process of training and testing the models. They were based on the raw executions available in Mattos *et al.* [41]. The first, referred to in this text as the *original* or the *big* dataset, was composed of a set of 2150 executions of length  $10^6$  iterations (each iteration was an observation), generated from 184 instances of three optimization problems. In total, it had  $2.15 \times 10^9$  iterations. The second dataset, on the other hand, was composed of samples of the executions in the original dataset, specifically the iterations where changes in the minimum occur and those at every  $10^4$  iterations. This dataset is referred to as the *sample* or the *small* dataset and had 202 479 iterations.

The executions that composed both datasets originated from the benchmark instances listed in Table 1, where 45 were from the problem SCP, 132 from QAP, and 7 from PMP. Each instance was obtained from one of two possible repositories and was represented by 10, 20 or 30 executions in the original dataset and 10 in the sample dataset.

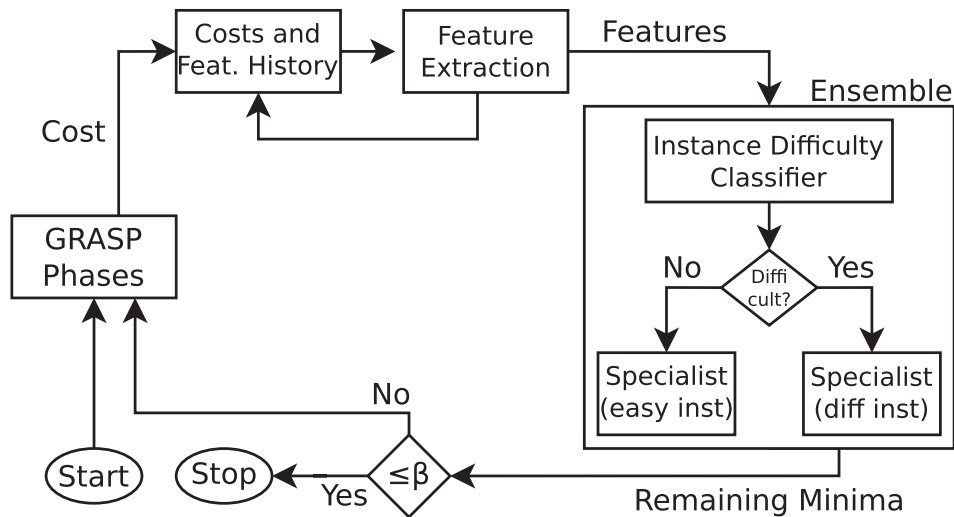


FIGURE 3. GRASP loop containing the stopping rule.

The SCP and the PMP were sourced from the OR-Library [6] and originated from Beasley [4, 5]. That repository also provides the costs for the best known solutions in the case of PMP. Meanwhile, the QAP instances are available in the QAPLIB [11] and come from a multitude of works. In this case, the sources and the best known solutions are listed on the library’s website.

As for the GRASP implementations used to produce the executions, there were three. The one for QAP is the same as that used in Oliveira *et al.* [47], while the implementation for the Set  $k$ -Covering Problem (SCP) is from Pessoa *et al.* [52]. In the case of the latter, the code was run with  $k = 2$  so that the best known solutions provided by the work from which it originated could be used when necessary. For the PMP problem, an implementation [39] was created specifically for the present work, and the small number of PMP instances resulted from the fact that this implementation had poor time performance. It is important to mention that the three GRASP implementations were always executed with RCL size 0.2 (the way this value was used varied depending on the GRASP implementation).

Regarding the features that represent each iteration, they are listed in Table 2, where a brief description is also provided. In general, they cover three aspects of an execution: time, represented by the feature *iteration*; instance characteristics, indicated by *symmetric* and *sparsity*; and cost statistics, represented by the remaining features. Together, they try to represent an iteration in a manner that is generic enough to allow generalization across instances of different problems. The exceptions to that are *symmetric* and *sparsity*, which are not as generic because some problems may have their instances described by only one type of matrix (for example, the SCP instances employed in this work are always represented by non-square matrices).

In addition to training features, the dataset also included the target variables *remaining\_mins* and *difficulty*. *remaining\_mins* acted as the main target of the dataset (the value to be estimated by the ensemble) and represented the approximate number of future minima to be found in the GRASP execution if it were run until an optimal solution was found. In practice, this approximation was calculated by observing a limited number of iterations in the future and counting the number of improved solutions in it. In Mattos [38], that interval had an upper limit at iteration  $10^6$ , which represents a limitation, for the reasons mentioned in Appendix A. An alternative would be the use of the interval  $[i + 1, i + 10^6]$ , for a given iteration  $i$ , similarly to what is done by Ribeiro *et al.* [57] to evaluate the accuracy of their stopping rule. However, that approach could experience instability over time because the number of remaining minima might increase from one iteration to another as a result of the change in the evaluated interval.

TABLE 1. List of instances of each difficulty per problem.

Prob.	Diff.	Instances
QAP	Diff.	chr20b, chr22a, chr22b, lipa50a, lipa60a, lipa70a, lipa80a, lipa90a, sko100a, sko100b, sko100c, sko100d, sko100e, sko100f, sko42, sko49, sko56, sko64, sko72, sko81, sko90, ste36a, ste36c, tai100a, tai100b, tai30a, tai35a, tai35b, tai40a, tai40b, tai50a, tai50b, tai60a, tai60b, tai80a, tai80b, tho40, wil100, wil50
	Easy	bur26a, bur26b, bur26c, bur26d, bur26e, bur26f, bur26g, bur26h, chr12a, chr12b, chr12c, chr15a, chr15b, chr15c, chr18a, chr18b, chr20a, chr20c, chr25a, els19, esc128, esc16a, esc16b, esc16c, esc16d, esc16e, esc16g, esc16h, esc16i, esc16j, esc32a, esc32b, esc32c, esc32d, esc32e, esc32g, esc32h, esc64a, had12, had14, had16, had18, had20, kra30a, kra30b, kra32, lipa20a, lipa20b, lipa30a, lipa30b, lipa40a, lipa40b, lipa50b, lipa60b, lipa70b, lipa80b, lipa90b, nug12, nug14, nug15, nug16a, nug16b, nug17, nug18, nug20, nug21, nug22, nug24, nug25, nug27, nug28, nug30, rou12, rou15, rou20, scr12, scr15, scr20, ste36b, tai10a, tai10b, tai12a, tai12b, tai15a, tai15b, tai17a, tai20a, tai20b, tai25a, tai25b, tai30b, tai64c, tho30
SCP	Diff.	scp41, scp410, scp42, scp43, scp44, scp45, scp46, scp47, scp48, scp49, scp51, scp510, scp52, scp53, scp54, scp55, scp56, scp57, scp58, scp59, scp61, scp62, scp63, scp64, scp65, scpa1, scpa2, scpa3, scpa4, scpa5, scpb1, scpb2, scpb3, scpb4, scpb5, scpc1, scpc2, scpc3, scpc4, scpc5, scpd1, scpd2, scpd3, scpd4, scpd5
PMP	Diff.	pmed10, pmed2, pmed3, pmed4, pmed5
	Easy	pmed1, pmed6

TABLE 2. Features available in the dataset.

N	Feature	Description
1	iteration	Number of the current iteration in its execution.
2	min_changes	Number of minimum changes.
3	skewness	Skewness of the cost value distribution.
4	kurtosis	Excess kurtosis of the cost value distribution.
5	z-min	Standardized minimum cost.
6	z-max	Standardized maximum cost.
7	minmaxdist	Difference between stdmax and stdmin.
8	symmetric	Instance matrix is symmetric or not.
9	sparsity	Sparsity of the instance matrix.
10	best_iter	Iter. where the most recent min. change happened.

For those reasons, the approach taken in the present work was different. Firstly, the probability of improvement  $p_{\text{improv}}$  was calculated by counting the number of remaining minima in the interval  $[i + 1, 2 \times 10^6]$  and dividing it by  $2 \times 10^6$  (the size of the raw execution<sup>1</sup>). The fixed upper limit for the evaluation interval and the fixed denominator in the division guaranteed the stability of the probability. Then, after that, the value of *remaining\_mins* for a given iteration was the result of  $p_{\text{improv}} \times 10^6$ . Considering that the executions in the dataset included only the first  $10^6$  iterations, it guaranteed that the counting interval always had a length of at least  $10^6$  iterations. As for the variable *difficulty*, its value was defined as presented in Appendix A.

#### 4.1.2. Dataset characteristics

The dataset analysis conducted in this work is partially available in Appendix A, where some insights about the instances that compose the dataset were provided in the context of showing limitations of existing stopping rules. In the present section, the focus is on the correlation between the features and the targets, as well as some

<sup>1</sup>There are 4 raw executions that are smaller. But they are still longer than  $10^6$  iterations.

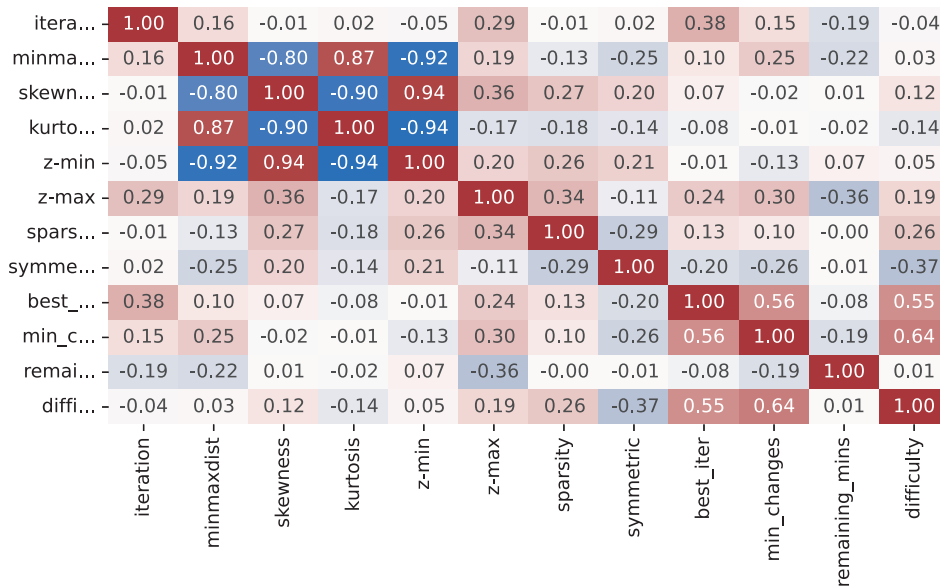


FIGURE 4. Correlation between the dataset features and targets.

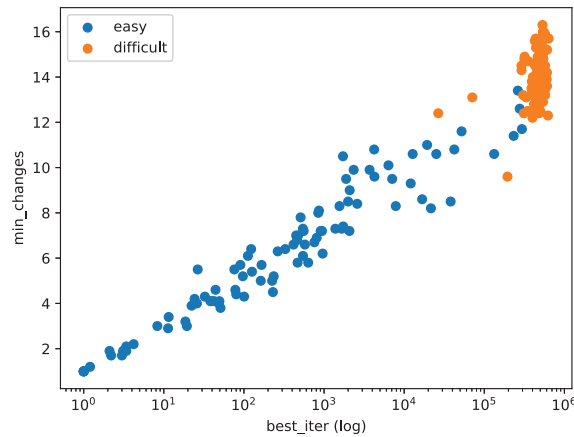


FIGURE 5. Separation of *difficulty* based on *best\_iter* and *min\_changes*.

related insights. That being said, in Figure 4, a heat map showing the linear correlation (Pearson Correlation Coefficient) is presented, and two regions are of particular interest. Firstly, the image shows that there is a very strong (negative or positive) correlation between the features *minmaxdist*, *skewness*, *kurtosis* and *z-min*. That result is expected because, whenever *z-min* decreases, *minmaxdist* increases, and *skewness* and *kurtosis* are also expected to be impacted due to the increase in size that the left tail experiences.

The other region of interest involves the features *best\_iter* and *min\_changes*, and the targets *difficulty* and *remaining\_mins*. In this case, the two features have a high correlation with *difficulty*, while only *min\_changes* has a non-negligible – but low – correlation with *remaining\_mins*. If the other features are observed, *remaining\_mins* also presents a low correlation with *iteration* and *minmaxdist*, and a moderate correlation with *z-max*.

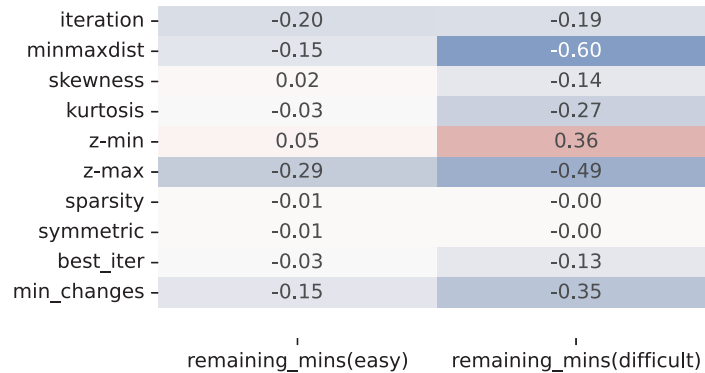


FIGURE 6. Correlation per difficulty.

TABLE 3. Mutual information between the features and *remaining\_mins*.

Feature	MI	Feature	MI
best_iter	1.26	z-max	0.25
z-min	0.52	skewness	0.22
iteration	0.37	kurtosis	0.21
minmaxdist	0.36	min_changes	0.21
sparsity	0.35	symmetric	0.02

Returning to the correlation between *difficulty*, *best\_iter* and *min\_changes*, if the perspective of the last iteration of the executions is taken, the two features make it possible to separate the instances linearly by difficulty in a satisfactory way, with little overlap, as can be deduced from Figure 5. In that figure, the values for those two features are obtained at the end of each execution of each instance (limited to the executions present in the small dataset), and those belonging to the same instance were averaged together, providing a single point to represent each instance. That figure also shows that easy instances usually reach their best solution in earlier iterations and with fewer changes to the minimum, while difficult instances usually behave in the opposite way. It provides an additional perspective to a similar observation made in Appendix A.

Given that difference in behavior, investigating the correlation between the features and *remaining\_mins* separated by instance difficulty is pertinent. In Figure 6, a heat map showing that information is provided, and it is possible to see that, in difficult instances, the correlation ranged from low to high, and only the features related to instance characteristics (*symmetric* and *sparsity*) showed negligible results. On the other hand, in easy instances, only 4 features showed non-negligible values.

In order to capture not only linear correlation but also non-linear dependence, this work also employed a measure called Mutual Information (MI). That method was applied to calculate the mutual dependence between each feature (independently of others) and the target *remaining\_mins*. It returns a non-negative number where the higher it is, the higher also is that dependence. The results are provided in Table 3, where it is possible to see that *best\_iter* has a high relation to the target, with a score that is more than twice the score obtained by *z-min*, which comes second. It is also pertinent to point out that, while *sparsity* had close to zero linear correlation with the target, it is the fifth in mutual information. On the other hand, *symmetric* continues to show negligible relevance.

TABLE 4. Hyperparameters optimized and value ranges used.

Algorithm	Hyperparameter	Value range
Random Forest Classifier	n_estimators	10–100
	max_depth	5–30
	min_samples_leaf	10–25
	n_neighbors	1–100
$K$ -Nearest Neighbor	weight	uniform, distance
	$p$	1–2
	n_clusters	2–100
$K$ -Means + R.F. Classif.	n_estimators	10–100
	max_depth	5–30
	min_samples_leaf	10–25
	n_estimators	10–100
Random Forest Regressor	max_depth	5–30
	max_features	2–5
	min_samples_leaf	10–25

## 4.2. Evaluation approaches

### 4.2.1. Evaluation from three perspectives

In order to evaluate the proposed ensemble model, the present work relied on three different tests. The first consisted of an exploratory experiment that focused on identifying prominent setups for the ensemble. It also examined the possibility of producing a good model trained on a small number of instances from the sample dataset. The second test used the most prominent model setup from the first test and used cross-validation to have a better idea of the performance of the stopping rule if the entirety of the available instances were used to produce a model. Lastly, the third test checked what this paper calls the interproblem generalization, where models were trained on a group of problems and used to produce estimates for an unknown problem.

In the approach taken in the first test, three classification algorithms were evaluated. They were the Random Forest Classifier (RFC), the  $K$ -Nearest Neighbors (KNN), and a combination of  $K$ -Means and Random Forest Classifier (KM+RFC). The reason for the first two lies in the fact that they are simple techniques and fast to train (the Random Forest model is also fast to estimate values). KM+RFC, on the other hand, was a result of trying to use  $K$ -Means in a classification-*via*-clustering setting [59] in the early stages of this work. The addition of the RFC came from the necessity of ways for dealing with problems that occurred when two centroids shared the same position (something that could happen when a large number of centroids were used). In this case, the role of the classifier was to learn how to classify iterations based on the relabeled training data produced by the clustering procedure. That relabeling consisted of performing the clustering and attributing to all observations in a given cluster the dominant class among them (*easy* or *difficult*).

For each of the models in the ensemble, the hyperparameters used in the hyperparameter optimization (HPO) can be seen in Table 4. Their meaning can be found in the documentation of the respective libraries used (the libraries are mentioned in Section 4.3 and any hyperparameter not listed in the table was kept at its default value). The value ranges were chosen based on early tests and, in some cases, were influenced by the values used in Mattos [38]. The table also includes those for the regressor models, which were Random Forest Regressors (RFR). That type of model was chosen because it is fast to train and to produce predictions, characteristics that are convenient when performing HPO.

In addition to algorithm settings, the optimizer also chose the features and instances used for training. In the features' case, two sets were evaluated, and the optimizer was able to select any subset within a given set. The first set was composed of all features listed in Table 2, while the second excluded the feature *iteration*. A third possibility was also tested, where *iteration* was only excluded in the model trained on easy instances. Given

TABLE 5. Feature normalization bounds.

Feature	LB	UB
iteration	1	1 000 000
min_changes	0	100
skewness	-5	5
kurtosis	-5	5
minmaxdist	0	20
stdmin	-5	0
stdmax	0	5
symmetric	0	1
sparsity	0	1
best_iter	0	1 000 000

the fact that some of the machine learning algorithms were not tree-based, normalization of the data was also performed (between 0 and 1). The bounds used for that purpose can be seen in Table 5.

Regarding the selection of the training instances, it was done among 20, where 10 were easy and 10 were difficult. The specific difficult instances were scp47, scp55, scpa2, tai30a, tai35a, tai40a, tai50a, pmed2, pmed5 and pmed10, and the easy instances were had12, scr20, ste36b, chr20a, had18, bur26d, pmed1, tai12a, tai20b and tai15a. Some of them were chosen because they were used in works mentioned in Section 2, while the others were drawn at random. All instances not used for training were used exclusively for testing. The only model where instance selection was not performed was the classifier.

For the second test, the evaluation approach consisted of a stratified 10-fold cross-validation where the stratification was based on difficulty. In other words, in each fold, the number of easy instances was approximately the same as the number of difficult instances. This test used the type of classifier, the feature set (among the three options mentioned earlier), and the HPO optimizer environment (see Sect. 4.2.3) of the most prominent ensemble of the exploratory experiment. It also included some changes to the value ranges of some hyperparameters, influenced by the results observed in that first test. Given the fact that performing instance selection would not be compatible with cross-validation, the HPO did not perform it, and the model produced at each fold was trained on all the expected training data for that fold.

Finally, interproblem generalization was evaluated through a strategy known as *leave-one-domain-out* (LODO). In that approach, the number of trained models is equal to the number of domains and, for each model, a different domain is used for testing and all the others for training. As such, three different ensembles were created. Each was trained on data from two optimization problems and tested on a third problem that was not present in the training data. Regarding HPO, this approach employed it individually for each ensemble. The hyperparameters that were optimized, as well as the value ranges, were the same used during the cross-validation.

In these three tests, the models were trained using the small dataset defined in Section 4.1 and the evaluation was performed using a set of metrics defined in Section 4.2.2. The performance of the models on the complete executions of the original (big) dataset was also verified. In those cases, a comparison with the results of the CDF of the gamma distribution (fitted to non-reflected costs) was also provided, as it is the most prominent estimator from Appendix A. In the second and in the third tests (cross-validation and interproblem generalization), the comparison also involved the CDF of the normal distribution and the two machine learning models (trained on a single instance each) of Appendix A. In the case of the CDF of the gamma distribution, it is important to mention that, in all executions, the estimates were only calculated in four situations: at iterations where the minimum changed; at every iteration from iteration 2 to 100; at every 100 iterations from iteration 200 to 10 000; and at every  $10^5$  iterations from iteration  $10^5$  to  $10^6$ . That approach was taken because the calculation of the MLE at every iteration would be very costly. Considering that the distribution of cost value is not well-defined

at early iterations, the comparisons included only the results from iteration 100 and above. As for the details of the HPO process, Section 4.2.3 presents its settings.

#### 4.2.2. Metrics

In Mattos [38], the root mean squared error (RMSE) was used as an evaluation metric. However, while it is sensitive to large divergences, it is not as sensitive when it comes to detecting stagnation in the estimates, which end up shadowed by an eventual explosion of the metric value if large divergences occur. In addition, it does not measure whether a target threshold  $\beta$  is being reached at the proper moment. With the inclusion of a classification model in the ensemble architecture proposed in Section 3, that metric is also inadequate to evaluate that classifier.

For these reasons, in the present work, a multifaceted evaluation was applied, based on multiple metrics. In that approach, the RMSE kept its role in detecting large divergences; however, stagnation was better visualized through the mean absolute error (MAE) when evaluating a full execution, because the large number of iterations reduces the impact of large divergences that last for a relatively small number of iterations. The classifier, on the other hand, was evaluated using the F1 score and the Macro F1 score (referred to in this paper as MF1), which are traditional metrics for classification tasks.

Regarding the evaluation of how well an estimator performs in making an execution stop at the expected moment, a simple custom metric, which was referred to as Similarity at Threshold (SAT), was employed. In that metric, the idea is to compare the iteration  $i_e$ , where a given threshold is reached in the curve of the estimates, to the iteration  $i_t$ , where that threshold is reached in the curve of the targets. However, given the fact that the absolute difference between those iterations can range from zero to a large number, the approach taken was to use some form of similarity. The resulting metric, for the case where a single threshold is evaluated, can be seen in equation (4). In that equation, the smallest among those two values is simply divided by the largest, resulting in a value that is guaranteed to fall in the range between zero and one. In the case where  $m$  thresholds are evaluated at once, the metric takes the general form presented in equation (5), here referred to as Mean Similarity at Threshold (MSAT), where  $i_e^{(1)}$  and  $i_t^{(1)}$  are the iteration numbers related to the first threshold (in the set of thresholds being evaluated),  $i_e^{(2)}$  and  $i_t^{(2)}$  refers to the second threshold, and so on. In the tests reported in Section 4.4, the MSAT was used to evaluate the performance at the thresholds 100 000, 10 000, 1000, 100, 10 and 1 (they could be  $10^{-1}$ ,  $10^{-2}$ ,  $10^{-3}$ ,  $10^{-4}$ ,  $10^{-5}$  and  $10^{-6}$ , if the targets were probabilities).

$$\text{SAT} = \frac{\min(i_e, i_t)}{\max(i_e, i_t)} \quad (4)$$

$$\text{MSAT} = \frac{\frac{\min(i_e^{(1)}, i_t^{(1)})}{\max(i_e^{(1)}, i_t^{(1)})} + \dots + \frac{\min(i_e^{(m)}, i_t^{(m)})}{\max(i_e^{(m)}, i_t^{(m)})}}{m}. \quad (5)$$

To demonstrate the MSAT in practice, two illustrative executions identified as *Execution 1* and *Execution 2* can be considered. In *Execution 1*, the curve of remaining minima reaches the thresholds at iterations 7, 120, 27 000, 78 000, 720 000, and 720 000, respectively. In *Execution 2*, this happens at iterations 12, 420, 1800, 137 000, 248 000, and 534 000, respectively. The behavior of those two executions can be visualized in Figure 7. The resulting MSAT is calculated as  $7/12 + 120/420 + 1800/27000 + 78000/137000 + 248000/720000 + 534/720 \approx 0.43$ , where the approximate similarities at threshold are, respectively, 0.58, 0.29, 0.07, 0.57, 0.34, and 0.74. This shows that the similarity at threshold  $\beta = 1000$  is the main responsible for reducing the MSAT.

As for the limitations of those custom metrics, one resides in the fact that it is necessary to develop an additional rule to treat thresholds that are not reached in one of the curves. In those cases, the simplest approach (which was the one taken in the present work) is to attribute zero to the similarity at those particular thresholds. However, in the MSAT, there might be occasions where different approaches, such as a less harsh penalty, might be desirable. In addition to that, the situation where a threshold is not reached by both curves must also be treated, with a suitable approach (which was used in this work) being to consider a similarity of 1 at that threshold, because it is not expected to be reached and should not penalize the final result.

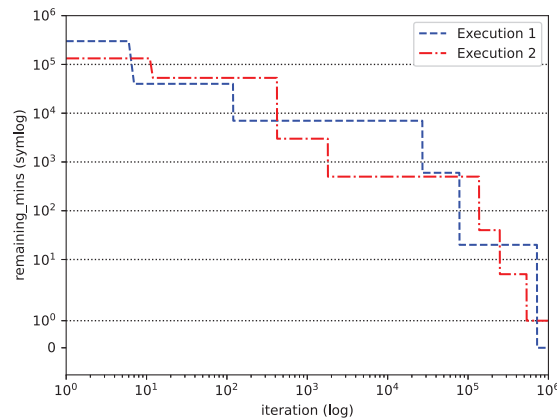


FIGURE 7. Example of two illustrative executions reaching different thresholds of remaining minima.

Another limitation resides in some situations where the distance between  $i_e$  and  $i_t$  is different in orders of magnitude, such as  $i_e = 7$  and  $i_t = 34$ , because the SAT value might end up being undesirably small, as in  $7/34 \approx 0.2$ , even though the distance in number of iterations is merely 27, which is negligible. This problem can be mitigated in the MSAT by applying weights to each SAT value and reduced weights to the thresholds where this kind of problem is prone to happen (mainly, 100 000 and 10 000; or  $10^{-1}$  and  $10^{-2}$ , if the targets are probabilities). Ignoring those potentially problematic thresholds is also an option.

In addition to the use of all metrics in their standard form (mainly to show training results), in tests that evaluated each execution individually, the values reported were the averages obtained across the set of executions. Seeking to highlight these occurrences, in cases where the reported value is an average, the metric name was written with the symbol  $\mu$  as subscript, as in  $\text{MSAT}_\mu$ .

Finally, it is important to mention that some other similarity metrics that could be used to replace the SAT and the MSAT are presented in Fink and Pratt [22]. The most prominent are the basic similarity defined by  $s(i_e, i_t) = 1 - |i_e - i_t| / (|i_e| + |i_t|)$  and its aggregate versions called *mean similarity* and *root mean square similarity*, given by  $(\sum_{j=1}^m s(i_e^{(j)}, i_t^{(j)})) / m$  and  $\sqrt{((\sum_{j=1}^m s(i_e^{(j)}, i_t^{(j)})^2) / m)}$ , respectively. The reason why they were not used was because the function  $s(i_e, i_t)$  is non-linear and produces higher values than the SAT, which could cause bad similarities to be interpreted less negatively than desired. Another option provided by that same work is the *peak similarity*, which is defined as  $ps(i_e, i_t) = 1 - |i_e - i_t| / (2 \times \max(|i_e|, |i_t|))$  and could be used to replace the SAT. However, because of the denominator of the division, it produces even higher values and has a lower bound at 0.5 in the particular case where one of the parameters is zero.

#### 4.2.3. Hyperparameter optimization

The hyperparameter optimizer used in this work consisted of a GA that was set up using a multi island architecture in order to allow the optimization to take advantage of the parallelism offered by contemporary computers. The parameter choices were partially influenced by Luque and Alba [37] and Belding [7]. The GA was composed of 4 or 8 islands, each with a population size of 50 individuals, totaling from 200 to 400 individuals across all islands. The two possible numbers of islands were used in the exploratory experiments, while, in the cross-validation, only 4 were used. The reasons for the smaller number of islands in those cases were different. In the first, it was related to the environment available at the time, while in the second, 4 threads were used by each individual to speed up their training time, making 4 islands a safe setup for a processor with 16 cores, as each island would be training one individual at a time.

Moving to other parameters, in the exploratory experiments, two combinations of migration gap and migration phases were tested. The first was a gap of 25 generations and 20 migrations, while the second was 50 and 10, respectively. In both cases, 500 generations were executed. As for the migration topology, two were evaluated in that same test. The first was more aimed toward diversification and consisted of each island receiving the best individuals from two randomly chosen islands (one from each). Before inserting them, the local population was restarted. The second topology focused on intensification: after each migration phase, the population of each island was restarted, and the best individual of each island was inserted.

In addition to those settings, the other parameters pertinent to the GA were the offspring size, which was set to 30; the mutation rate and degree, which were set to 50% and 20%; the crossover method, which was uniform; and the selection method, which was the 3-way tournament and was applied to select parents for crossover and individuals to be transferred between generations. In all generations, only the best individual was guaranteed to be transferred to the next generation. However, in Sections 4.4.2 and 4.4.3, that number was increased to 5.

As described in Section 4.2, the optimization included the training instances, the training features, and the hyperparameters of the model. However, in the case of instance selection, it was applied only to the model trained on a limited subset of the dataset because otherwise there would be too many possible combinations. It is also pertinent to mention that the value intervals presented in Table 4 are discrete, and no prior discretization was necessary because they already had that characteristic (for example, it is not possible to have 20.5 estimators).

Finally, the fitness function used is presented in equation (6). It is a result of the scalarization of the multiple metrics (objectives) mentioned in Section 4.2.2. The approach is similar to the weighted product method presented in Section 2, but without weights. One drawback that it has is that it can reach 0 if the MF1 reaches 1, because a perfect MF1 does not mean a perfect score in other metrics. However, that problem can be avoided by guaranteeing that the results of the operations between parentheses never reach 0. This can be achieved by having those variables subtracted from 1.0001 instead of 1. That modification had to be made in Section 4.4.3 because the MF1 reached 1 in one of the models.

$$-1 \times \text{MAE} \times \text{RMSE} \times (1 - \text{MF1}) \times (1 - \text{MSAT}_\mu). \quad (6)$$

### 4.3. Computational setup

The work presented in this paper was conducted on two kinds of computational platforms, one for software development and prototyping, and another for training the models. Although the development platform is not particularly relevant, training was conducted on server grade Ampere Altra processors, based on the ARM architecture, running at 2 GHz at Oracle Cloud and at Hetzner Cloud (likely a Q80-30, with up to 4 cores at Oracle and 16 cores at Hetzner). As for the libraries pertinent to the results of the experiments, they were NumPy [29] v2.0.2, SciPy [65] v1.15.1 (used for its statistical functions), Pandas [64] v2.2.3, Scikit-Learn [50] v1.2.2 and v1.6.0 (machine learning algorithms and metrics), Statsmodels [58] v0.14.4 (Q-Q plots), Matplotlib [32] 3.10.6, and Autorank [30] v1.3.0 (Critical Difference diagrams).

### 4.4. Results and analysis

#### 4.4.1. Exploratory experiments

Given the evaluation approach presented in Section 4.2, this section starts with the results obtained for the first test. In Table 6, each row represents one test ensemble that was subjected to HPO, and all of its values are truncated to the first or the second decimal place, as in all other tables in this section. In the first column, a numeric identification for the estimator is provided, and, in the second column, the classifier used at the top of each ensemble is presented. The third column covers the topology employed by the GA, where **A** means the one in which each island receives two migrants during the migration phase, and **B** represents the topology in which each island receives one migrant from each island. As for the fourth column, it covers the set of features available during the HPO process, with **M** being the set of all features, **N** being the set without the feature *iteration*, and **O** being the set where only the regressor trained on easy instances did not have the feature *iteration* available.

TABLE 6. Results for the training involving only the small dataset. Metrics followed by a  $\mu$  indicate that the values are averages of the results observed per execution.

Est.	Classifier	Topol.	Feat.	Training				Test				Fitness
				MAE	RMSE	MF1	MSAT $_{\mu}$	MAE	RMSE	MF1	MSAT $_{\mu}$	
1	KM+RFC	A	M	5148.7	36834.8	0.88	0.33	5192.4	37612.9	0.89	0.32	-14314849.8
2			N	5543.9	40078.4	0.89	0.40	10100.1	54929.5	0.88	0.37	-40798552.2
3			O	5112.6	37321.2	0.96	0.35	5425.7	38951.7	0.83	0.31	-23160447.8
4		B	M	5051.4	36761.0	0.91	0.39	5141.5	37701.0	0.89	0.31	-13588150.4
5			N	5428.6	39623.2	0.87	0.33	10056.6	55467.4	0.89	0.32	-40196689.6
6			O	5110.7	36998.3	0.90	0.35	5356.1	37838.8	0.84	0.34	-19824125.1
7	KNN	A	M	5136.7	37000.5	0.94	0.42	5150.3	37740.2	0.91	0.33	-11089648.0
8			N	5348.4	38548.6	0.99	0.38	9983.5	54104.3	0.90	0.35	-32071798.7
9			O	5129.7	37101.8	0.96	0.35	5436.0	37916.6	0.86	0.32	-18860701.7
10		B	M	5101.0	36910.3	0.94	0.42	5178.5	37819.4	0.91	0.34	-11045659.0
11			N	5321.0	38691.5	0.99	0.40	10134.8	54074.0	0.90	0.36	-32300919.7
12			O	5110.1	37229.4	0.96	0.36	5425.0	38015.0	0.86	0.33	-18717593.4
13	RFC	A	M	5052.3	36771.3	0.94	0.36	5129.5	37553.6	0.90	0.33	-11813138.0
14			N	10725.3	59757.7	0.96	0.25	9563.0	52533.5	0.87	0.27	-44910785.5
15			O	8027.2	37384.4	0.93	0.36	8080.4	38114.7	0.90	0.33	-20225528.5
16		B	M	5024.8	36733.0	0.97	0.40	5139.4	37746.3	0.90	0.36	-12028174.0
17			N	5377.7	39248.5	0.96	0.38	10003.0	54537.7	0.87	0.34	-43912959.9
18			O	8276.1	37710.8	0.93	0.36	8171.3	38377.5	0.90	0.34	-20515697.2

The results in that table show that, among the alternatives tested, the best fitness was obtained by ensemble 10, which had a KNN at its top and the feature set **M** available during the training. The GA used topology **B**. In addition to that model, the second and the third best (numbers 7 and 13, respectively) also employed the feature set **M** and achieved a fitness that was relatively close to that obtained by the best model. The second place was an ensemble that had a KNN at its top as well, but the GA topology was **A**. The third place also used that same topology but employed an RFC as its classifier. Given the fact that random forest models are also ensembles, it is important to mention that, in this section, they will not be referred to by that word, as it would cause confusion because the architecture that they are part of is also an ensemble.

In order to evaluate the specific hyperparameters used during the training of those models, Table 7 can be inspected. In that table, it is possible to see that no ensemble used all features during the training of its internal models, with the classifiers being those that relied on the highest number of features, while the regressors used only from 1 to 3. Furthermore, it is evident that the best and the second-best ensembles had very similar hyperparameter values, with differences only in the training instances and the training features of the RFR models.

Those regressors also stand out for another reason. In the case of the ones trained on easy instances, they were less complex than their counterparts trained on difficult instances, being shallower and having fewer trees. The one in the third-best ensemble, for example, had a *max\_depth* of 5 and only 10 trees. That simplicity motivated the inspection of the feature importance of each feature. For ensembles 10 and 7, it was 0.98 and 0.97 for the feature *iteration* in the regressors trained on easy instances, meaning that the other features had little impact on the estimates. In the regressors trained on difficult instances, the dominant feature was *minmaxdist*, achieving an importance of 0.97 in the best ensemble and 0.8 in the second best, where *iteration* reached 0.19. The ensemble that occupied the third position showed importances close to those obtained by the best model,

TABLE 7. Hyperparameters of the top 3 models. Feature names were replaced by their number according to Table 2.

Role	Hyperparameter	1st Place (10)	2nd Place (7)	3rd Place (13)
Classifier	training_features	1, 2, 3, 4, 8, 10	1, 2, 3, 4, 8, 10	1, 2, 8, 10
	n_neighbors	74	74	N.A.
	weights	uniform	uniform	N.A.
	$p$	manhattan	manhattan	N.A.
	n_estimators	N.A.	N.A.	13
	max_depth	N.A.	N.A.	6
	min_samples_leaf	N.A.	N.A.	23
Regressor (easy instances)	training instances	had12, had18, bur26d, tai12a, tai15a	had12, had18, bur26d, tai12a, tai15a	had12, ste36b, bur26d, tai15a
	training features	1, 3	1, 3, 9	1, 5, 9
	n_estimators	12	12	10
	max_depth	12	12	5
	max_features	5	5	3
	min_samples_leaf	10	10	10
Regressor (difficult instances)	training instances	tai35a, pmed5, pmed10	tai30a, tai40a, pmed5, pmed10	scp47, scpa2, pmed5, pmed10
	training features	2, 7	1, 3, 7	7
	n_estimators	65	65	67
	max_depth	14	14	21
	max_features	4	4	2
	min_samples_leaf	10	10	10

but, in the case of the regressor based on difficult instances, it was expected, given the fact that it was trained using a single feature (*minmaxdist*).

In order to clarify whether the results obtained by those ensembles are sufficiently good, the values of the other metrics must be checked. The  $MSAT_{\mu}$  obtained by them were 0.33 and 0.34, which are below the 0.37 reached by ensemble 2 in Table 6. This estimator, as well as the others that used the feature set  $\mathbf{N}$ , shows much higher MAE and RMSE, which indicates a stagnation of the estimates at some high values in a significant number of instances, a factor that can be confirmed through visual inspection. Figures 8a and 8b, for example, show extreme cases where the estimates stagnated at very high values. In Figure 8a, the estimates continue to fall over time, but the execution ends with them still far from the target. Meanwhile, in Figure 8b, a proper stagnation at around  $10^4$  expected minima is shown. However, it is important to mention that, in those figures, nearly all iterations of the executions that they are based upon are shown, while the small dataset only includes some reference points for each execution, as described in Section 4.1.

In both figures, the iterations range from 100 to  $10^6$  and the results for the CDF of the gamma distribution are also presented. The exclusion of the first few iterations was done because those values were used to compose the initial sample that was used to fit the distribution. Unlike [19], the parameters of the distribution were recalculated periodically, based on all previous iterations. In iterations where it was not calculated, the probability obtained in the prior iteration was replicated. The increased frequency in the calculation was done in order to make the comparisons fairer by allowing the distribution to incorporate new information not only about the frequencies, but also about those brought by changes in the minimum and in the maximum. With that said, the observation that can be made about the CDF from what is shown is that it can stagnate at relatively large values, and also that the estimates can fall, increase again, and continue at a high value until the end of the execution. Those behaviors were observed in multiple easy instances.

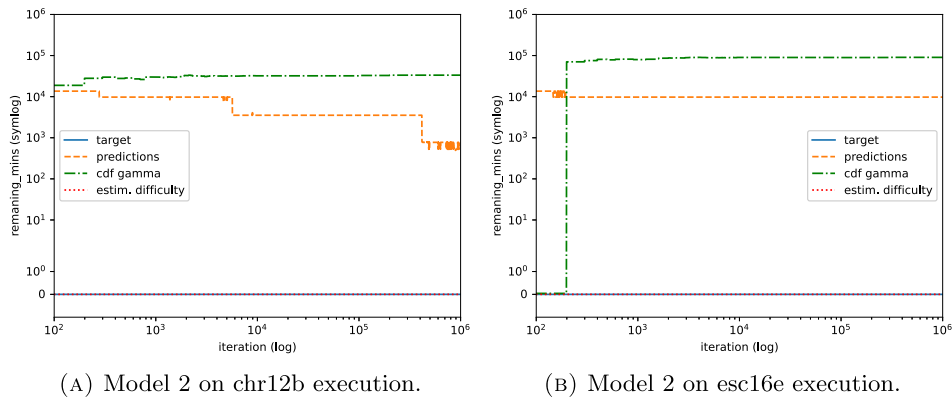


FIGURE 8. Stagnation of the model estimates on two easy instances. Results for the CDF of the gamma distribution are also included.

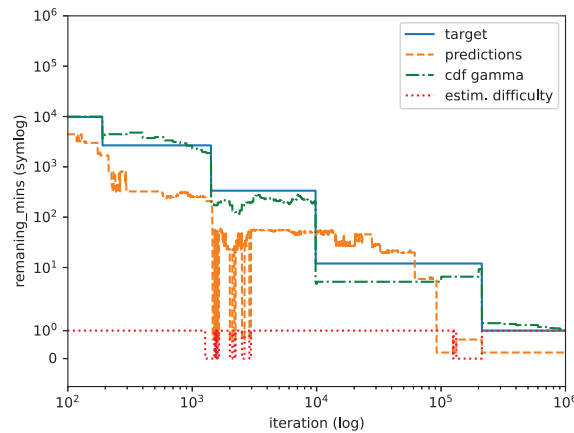


FIGURE 9. Example of temporary falls in the estimates. Results of the application of ensemble 10 on an execution of the instance scp63.

Proceeding with the matter of stagnation, all ensembles experienced it to some extent. The best three, for example, failed to reach the last one or two thresholds in easy instances on many occasions. In some cases, that was caused by the response being strongly tied to iteration number, which showed reduced flexibility, while in some other situations, the stagnation was due to difficulty misclassification.

An additional problem that can be reported is, in some sense, similar to that observed for the CDF, where the estimates fall drastically and rise again. In the case of the ensembles, the length of those falls can be short or very long, lasting from a few iterations to thousands. The result of this kind of problem is that, in some cases, an execution may stop much earlier than expected. An example of this kind of behavior is provided in Figure 9, where it is caused by difficulty misclassification. In that example, the overall fitting was also not good.

The ensembles not covered up to this point were those where the feature *iteration* was not available for use by the regressor trained on easy instances (those identified by an **O** in the column *Feat.*). The suppression of that feature was done in order to avoid the problem of having it achieve very high importance, which may result in estimates with very similar behavior on many instances. However, in the case of the ensembles 3, 6, 9 and 12, this approach ended up leading to a situation where *iteration* became dominant in the regressor trained on

TABLE 8. Evaluation of complete executions. Metrics followed by a  $\mu$  indicate that the values are averages of the results observed per execution.

Estimator	Easy test instances				Difficult test instances				All test instances			
	MAE $_{\mu}$	RMSE $_{\mu}$	F1 $_{\mu}$	MSAT $_{\mu}$	MAE $_{\mu}$	RMSE $_{\mu}$	F1 $_{\mu}$	MSAT $_{\mu}$	MAE $_{\mu}$	RMSE $_{\mu}$	F1 $_{\mu}$	MSAT $_{\mu}$
7	60.5	325.1	0.93	0.48	15.0	95.1	0.96	0.57	37.8	210.7	0.94	0.53
10	52.8	264.1	0.93	0.48	22.4	105.5	0.96	0.58	37.7	185.3	0.94	0.53
13	44.9	727.0	0.89	0.46	24.8	102.1	0.96	0.55	34.9	416.4	0.93	0.51
CDF	39 522.1	48 911.2	N.A.	0.51	371.2	428.1	N.A.	0.63	20 061.1	24 811.4	N.A.	0.57

difficult instances, with importance close to 1.0, and *minmaxdist* became dominant in the regressor trained on easy instances. The result of that was that the test MF1 scores were smaller. On many occasions, the behavior of the estimates was more unstable because of misclassifications, and in instances classified as difficult, the curve of the estimates was identical. Meanwhile, in the case of the ensembles 15 and 18, *iteration* was avoided, but the problem became stagnation, which is evidenced by the MAE and the RMSE values in the test set.

To conclude the exploratory experiments, the results of applying the three best ensembles to complete executions are provided and compared to the results of the CDF of the gamma distribution (the probabilities were converted to remaining minima by multiplying them by  $10^6$ ). In Table 8, each execution was evaluated separately, and the presented values are the average of what was reported by the metrics for the components of the set of executions. The exact number of instances and executions involved was 164 (only test instances) and 1710, respectively.

Those results show that, in the MSAT $_{\mu}$ , which is the most relevant metric, the ensembles that were trained on just 20 instances were unable to perform better than the CDF. Despite that, they were not far behind, with ensemble 10 being behind by approximately 0.03 and 0.05 in the easy and the difficult instances, respectively, and by 0.04 when all test instances are considered. As for the MAE $_{\mu}$  and the RMSE $_{\mu}$ , they confirm previous observations that the CDF can stagnate at high values. Out of 779 executions where the CDF's MAE was greater than 100, 668 belonged to a group of 68 easy instances, where the MSAT $_{\mu}$  was 0.46. However, it is important to mention that situations such as the one presented in Figure 9, where estimates fall and rise again, happened in many of those cases. That is a factor that explains why the MSAT $_{\mu}$  was not lower.

#### 4.4.2. Evaluation through cross-validation

In this test, the approach taken was to perform a stratified 10-fold cross-validation for an ensemble with some basic settings similar to those of the ensemble 10 from Table 6. However, some changes were made. In all random forest models, the *max\_depth* range considered in the HPO was 10 to 30, and *max\_features* was set to be the square root of the number of features. The number of migration phases was also reduced to 5, due to the number of models trained in each GA individual becoming several times higher because of the cross-validations. Additionally, instance selection was not performed, and the number of islands was changed to 4 in all cases. This change in the number of islands was done in order to allow the allocation of multiple threads to the models during their training phase.

The results obtained are presented in Table 9 and show that the cross-validation scores, in the four basic metrics, are close to the results reported in Table 6 for ensemble 10. The relevant difference lies in the fitness, where the associated cross-validation score was significantly higher (which is better, given the fact that the fitness function is from a maximization process). However, the standard deviation of the values used to calculate that score was also high, which indicates that the fitness observed at each fold varied considerably. In practice, it ranged from  $-1\,555\,854.8$  to  $-13\,170\,742.1$ , with  $-8\,148\,407.7$  as the median fitness.

Regarding the features and hyperparameters used in each of the ensemble's models, they are presented in Table 10. Once again, it is possible to see that the RFR trained on easy instances was shallow and had a small

TABLE 9. Cross-validation scores based on each metric.

Metric	Score	Std. Dev.
MAE	5274.7	237.0
RMSE	37 442.1	1013.3
MF1	0.93	0.03
MSAT <sub>μ</sub>	0.34	0.02
Fitness	-7 680 862.9	3 832 057.6

TABLE 10. Hyperparameters of the ensemble from Table 9.

Role	Hyperparameter	Values
Classifier	training features	1, 4, 7, 8, 9, 10
	n_neighbors	17
	weights	distance
	p	manhattan
Regressor (easy instances)	training features	1, 2, 3, 8
	n_estimators	11
	max_depth	10
	min_samples_leaf	22
Regressor (difficult instances)	training features	3, 5, 7, 9, 10
	n_estimators	15
	max_depth	18
	min_samples_leaf	18

TABLE 11. Evaluation of complete executions. Metrics followed by a  $\mu$  indicate that the values are averages of the results observed per execution.

Estimator	Easy instances				Difficult instances				All Instances			
	MAE <sub>μ</sub>	RMSE <sub>μ</sub>	F1 <sub>μ</sub>	MSAT <sub>μ</sub>	MAE <sub>μ</sub>	RMSE <sub>μ</sub>	F1 <sub>μ</sub>	MSAT <sub>μ</sub>	MAE <sub>μ</sub>	RMSE <sub>μ</sub>	F1 <sub>μ</sub>	MSAT <sub>μ</sub>
Ensemble	317.9	878.3	0.94	0.40	17.9	102.7	0.97	0.67	161.7	474.3	0.96	0.54
CDF(gamma)	33 551.4	41 418.6	N.A.	0.50	287.6	342.7	N.A.	0.65	16 223.3	20 020.9	N.A.	0.58
CDF(normal)	74 757.7	74 766.5	N.A.	0.28	1739.8	1814.1	N.A.	0.39	36 720.5	36 763.4	N.A.	0.34
RFR(scp42)*	9262.8	9390.6	N.A.	0.36	8.4	112.4	N.A.	0.62	4504.7	4620.2	N.A.	0.50
RFR(pmed1)*	3.8	110.5	N.A.	0.46	7.8	99.5	N.A.	0.44	5.9	104.7	N.A.	0.45

number of estimators, while the RFR trained on difficult instances was more complex. The former also included the feature *iteration* among its training features. However, despite achieving a high importance, in this case it was smaller than in the cases mentioned in Section 4.4.1, ranging from 0.78 to 0.82 (it was different in each fold). At the same time, in the model trained on difficult instances, the highest importance reached was approximately 0.45, by the feature *minmaxdist*.

The numbers in Table 9 are based on the small dataset and provide only a limited idea about the performance on complete executions. For that reason, as in the exploratory experiment, Table 11 shows the aggregate performance of the proposed method when applied to the executions of the big dataset individually. In this case, each execution was evaluated by the model from the fold where it figured in the test set, and the reported aggregate metrics were calculated over the entire set of executions.

TABLE 12. Ensemble and CDF results separated by instance difficulty. Metrics followed by a  $\mu$  indicate that the values are averages of the results observed per execution.

Instance difficulty	Better Estim. (MSAT)	Num. executions	Ensemble results				CDF results		
			$F1_\mu$	$MAE_\mu$	$RMSE_\mu$	$MSAT_\mu$	$MAE_\mu$	$RMSE_\mu$	$MSAT_\mu$
Easy	CDF	523	0.94	390.6	1327.0	0.36	52101.0	65884.9	0.67
	Ensemble	507	0.94	242.9	415.3	0.43	14416.4	16180.2	0.31
Difficult	CDF	512	0.97	15.4	117.9	0.64	36.1	90.1	0.77
	Ensemble	608	0.97	20.1	89.9	0.70	499.4	555.4	0.55

The results show that, when all executions were considered, the CDF of the gamma distribution achieved the highest  $MSAT_\mu$  and the ensemble ranked second, with a difference of 0.04. The same difference was observed between the second and the third place, which was occupied by the RFR model trained on executions of the instance scp42. However, that model cannot be reliably compared to the ensemble or any other estimator because its results do not include scp42 executions, as they were present in the training set. A similar situation applies to the RFR model trained on pmed1 executions, because no pmed1 execution was included in the model's evaluation (the "\*" in an estimator's name was used to indicate that situation). Despite that, these results were presented together for completeness and ease of comparison.

If the focus is shifted to the difficult instances, that table indicates that the ensemble showed a slightly better  $MSAT_\mu$  than that of the CDF of the gamma distribution. The worst performance in the complete dataset is explained by the results in easy instances, where the proposed method reached an  $MSAT_\mu$  of only 0.4, being 0.1 behind the CDF.

Investigating these results further, Table 12 shows that, in the case of executions from difficult instances, the ensemble showed an equal or better  $MSAT$  in approximately 54% of them, and, in the cases where it was behind the CDF of the gamma distribution, the  $MSAT_\mu$  was still above 0.6. Meanwhile, in the case of executions from easy instances, the ensemble provided an equal or better  $MSAT$  in approximately 49% of them. However, in the other 51%, the difference in  $MSAT_\mu$  was greater than 0.3. If the  $MAE_\mu$  and  $RMSE_\mu$  of the ensemble are taken into consideration, it becomes evident that it was caused by the estimates stagnating at large values. The lower  $F1_\mu$  (if compared to the case of difficult instances) also suggests that difficulty misclassification was part of the problem.

In Table 11, the high  $MAE_\mu$  and  $RMSE_\mu$  observed for the CDF estimators and for the RFR trained on scp42 executions also indicate the occurrence of stagnation of estimates, but mainly on easy instances. In the case of the RFR model, that situation is justifiable because it was not trained on easy instances. On the other hand, in the case of the CDF of the normal distribution, it presents a comparatively high  $MAE_\mu$  and  $RMSE_\mu$  also in difficult instances, a factor that contributes to its last position under the  $MSAT_\mu$  metric, as it indicates that it often stagnates in executions of that category of instance as well. As for the RFR model trained on pmed1 executions, the low  $MAE_\mu$  and  $RMSE_\mu$  are misleading, appearing to indicate a good performance. However, they are caused by the model starting to estimate 0 (or a value close to that) early in the execution. That often leads to relatively small divergences for most of the execution, and, consequently, large divergences in early iterations end up diluted because of the large size of the execution.

Considering that the training of the ensemble (based on the small dataset) was performed through cross-validation, it is pertinent to verify the cross-validation score also on complete executions. The results in Table 11 are expected to be close to such scores, but they slightly differ because the number of executions per fold is not always equal (some instances had more than 10 executions). Given that situation, in Table 13, the average of the  $MSAT_\mu$  per fold is presented. The focus on that metric is justified by the fact that it better describes the estimator's performance in their intended role (supporting the decision on whether to stop the metaheuristic).

TABLE 13. Average of the  $MSAT_{\mu}$  per fold.

Evaluation instances	Difficulty	Ensemble	Estimator			
			CDF (gamma)	CDF (normal)	RFR (scp42)	RFR (pmed1)
All Instances	Both	0.5437	0.5807	0.3434	–	–
	Easy	0.4006	0.5003	0.2822	–	–
	Difficult	0.6759	0.6555	0.3938	–	–
scp42 and pmed1 removed	Both	0.5431	0.5812	0.3454	0.5003	0.4579
	Easy	0.4016	0.5029	0.2829	0.3623	0.4663
	Difficult	0.6740	0.6536	0.3973	0.6276	0.4494

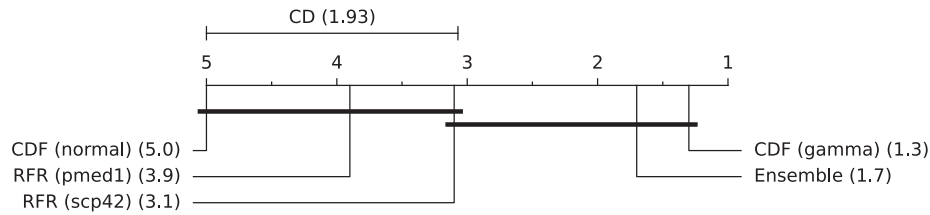
Those results are shown for the entire original dataset and also for a subset where instances scp42 and pmed1 are removed. This subset is used to allow for a reliable comparison between all estimators.

The results in that table, which were truncated at four decimal places for better clarity, do not provide new information compared to Table 11, except for the fact that no significant differences were observed in the first three estimators after the removal of the two aforementioned instances. However, the comparison between all estimators allows the verification of the statistical significance of the results, which shows if a given method is indeed better than the other. That procedure is pertinent because some scores were relatively close, like those for the ensemble and for CDF of the gamma distribution.

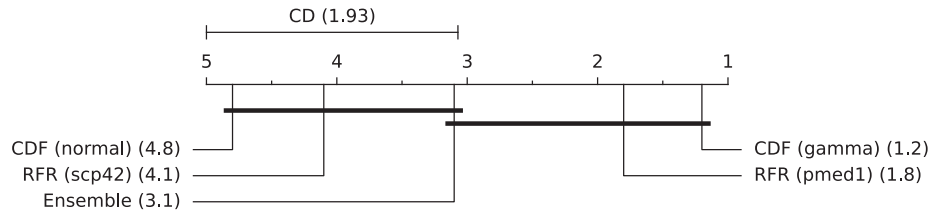
For that purpose, the present work used the Friedman test followed by the Nemenyi test, covered in Demšar [17]. It starts by conducting the Friedman test, whose null hypothesis states that all estimators have similar performance and that their average ranks should be equal. The alternative hypothesis, on the other hand, says that there is at least one estimator with a performance that differs. With a significance level  $\alpha = 0.05$ , the null hypothesis can be rejected if the  $p$ -value of the test is lower than  $\alpha$ . In the case of the  $p$ -values for the three types of datasets, they were approximately  $2.21 \times 10^{-7}$  for the dataset composed of only easy instances,  $1.91 \times 10^{-6}$  for the one with only difficult instances, and  $1.35 \times 10^{-7}$  for the complete dataset, which covers both difficulties. Given these results, the null hypothesis was rejected in all cases, which means that there was at least one estimator with a performance that differed from the others.

Once the null hypothesis is rejected, the Nemenyi test is used to conduct a pairwise comparison, where the estimators have their performance checked against each other. In order to show those results, Critical Difference (CD) diagrams were used. That kind of diagram shows the average rank of each estimator (based on their performance on each fold) and connects estimators with a black line if the difference between their results is not statistically significant. For any given pair of estimators, they are considered significantly different if their average ranks differ by at least the value of the CD (which depends on the number of estimators and datasets – in this case, folds); otherwise, the difference is statistically insignificant. In Figure 10a, the result for the Nemenyi test on the estimators evaluated on all executions is provided and shows that the CDF of the gamma distribution ranks first among the test estimators, with an average rank of 1.3. In second comes the ensemble, with 1.7. That figure also indicates that the difference between the results of those two estimators and between them and the RFR trained on the instance scp42, are not statistically significant, even though the average rank of that RFR model was 3.1, which is almost twice that of the ensemble.

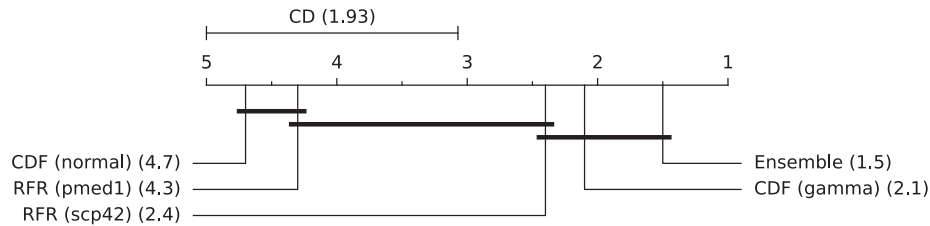
In Figure 10b, the evaluation was limited to executions from easy instances, and the first two positions were taken by the CDF of the gamma distribution and the RFR trained on the instance pmed1, with average ranks smaller than 2. The ensemble ranked third, and the difference between its results and the results of the first two estimators was not statistically significant (the difference when compared to the worst performing estimators was also insignificant). The next comparison is provided by Figure 10c, where only executions from difficult instances are considered. It places the ensemble in the first position, but the CDF of the gamma distribution



(A) Results when easy and difficult executions are considered.



(B) Results when only easy executions are considered.



(C) Results when only difficult executions are considered.

FIGURE 10. Critical Difference Diagrams showing the results of the Nemenyi test. Estimators connected by a horizontal bar are not significantly different in performance.

and the RFR trained on the instance scp42 come close after. The difference between the results of those three estimators is, once again, statistically insignificant.

#### 4.4.3. Evaluation through a Leave-One-Domain-Out strategy

Given the LODO strategy and the three optimization problems represented in the dataset, three combinations were tested. The first with QAP and PMP instances in the training set, and SCP instances in the test set; the second with QAP and SCP instances involved in the training and PMP instances in the test set; and the third with SCP and PMP instances used for training and the QAP instances used for testing. The results are presented in Table 14 and are based only on the small dataset.

Each ensemble in that table was trained using HPO, where the number of migration phases was increased back to 10. For the one trained on the combination of QAP and PMP instances, the test MF1 score was 1.0, which means that the iterations of the testing SCP executions were correctly classified as coming from difficult instances in all cases. The testing  $MSAT_{\mu}$  and the fitness were also the highest. However, the fitness value was mainly influenced by the MF1 score, which led to a near zero value being present in the multiplications executed in the fitness function.

TABLE 14. Interproblem performance involving only the small dataset. Metrics followed by a  $\mu$  indicate that the values are averages of the results observed per execution.

		Training results				Test results				
Training problems	Test problem	MAE	RMSE	MF1	MSAT $_{\mu}$	MAE	RMSE	MF1	MSAT $_{\mu}$	Fitness
QAP+PMP	SCP	5368.7	36 278.3	0.96	0.31	5199.39	37 350.7	1.00	0.49	-9 759.5
SCP+QAP	PMP	5304.6	37 091.7	0.99	0.39	4964.9	34 883.9	0.94	0.42	-5 565 386.6
PMP+SCP	QAP	5260.8	36 923.6	0.81	0.45	5294.0	37 692.8	0.75	0.24	-37 215 163.3

TABLE 15. Results for the evaluation on complete executions. Metrics followed by a  $\mu$  indicate that the values are averages of the results observed per execution.

		Easy Instances				Difficult Instances				All Instances				
Training problems	Test problem	Estimator	MAE $_{\mu}$	RMSE $_{\mu}$	F1 $_{\mu}$	MSAT $_{\mu}$	MAE $_{\mu}$	RMSE $_{\mu}$	F1 $_{\mu}$	MSAT $_{\mu}$	MAE $_{\mu}$	RMSE $_{\mu}$	F1 $_{\mu}$	MSAT $_{\mu}$
QAP+PMP	SCP	Ensemble	N.A.	N.A.	N.A.	N.A.	6.2	80.2	1.00	0.60	6.2	80.2	1.00	0.60
		CDF (gamma)	N.A.	N.A.	N.A.	N.A.	7.3	73.9	N.A.	0.69	7.3	73.9	N.A.	0.69
		CDF (normal)	N.A.	N.A.	N.A.	N.A.	1976.6	2079.4	N.A.	0.28	1976.6	2079.4	N.A.	0.28
		RFR (scp42)*	N.A.	N.A.	N.A.	N.A.	7.3	107.5	N.A.	0.62	7.3	107.5	N.A.	0.62
		RFR (pmed1)	N.A.	N.A.	N.A.	N.A.	7.8	101.4	N.A.	0.45	7.8	101.4	N.A.	0.45
SCP+QAP	PMP	Ensemble	90.7	145.4	0.98	0.46	13.3	62.7	0.98	0.65	35.4	86.3	0.98	0.60
		CDF (gamma)	1572.4	1576.6	N.A.	0.32	124.4	161.5	N.A.	0.57	538.1	565.8	N.A.	0.50
		CDF (normal)	10 351.8	10 353.9	N.A.	0.20	759.8	811.2	N.A.	0.35	3500.3	3537.7	N.A.	0.30
		RFR (scp42)	15.0	103.1	N.A.	0.43	7.0	97.6	N.A.	0.62	9.3	99.2	N.A.	0.57
		RFR (pmed1)*	3.5	94	N.A.	0.54	7.3	91.5	N.A.	0.46	6.7	91.9	N.A.	0.47
PMP+SCP	QAP	Ensemble	6.6	92.1	0.75	0.42	6.4	85.3	0.99	0.62	6.6	89.9	0.82	0.48
		CDF (gamma)	34 843.5	43 028.4	N.A.	0.50	650.4	695.8	N.A.	0.63	23 836.1	29 400.8	N.A.	0.54
		CDF (normal)	77 360	77 369.1	N.A.	0.28	1671.3	1717.2	N.A.	0.54	52 994.5	53 015.4	N.A.	0.36
		RFR (scp42)	9636.5	9765.8	N.A.	0.36	9.9	121	N.A.	0.62	6537.5	6661	N.A.	0.44
		RFR (pmed1)	3.8	110.9	N.A.	0.46	8	99	N.A.	0.44	5.2	107	N.A.	0.45

The performance on the training set was lower, resulting from the model selection focusing only on the test set. This highlights a possible disadvantage of that exclusive focus and of using a multiplicative scalarization, because the excessive impact of the MF1 score might have led to the dismissal of estimators with better or more balanced training and testing MSAT $_{\mu}$ .

For the ensemble based on SCP and QAP instances, the training and the testing MSAT $_{\mu}$  were more equilibrated, and the testing MAE and RMSE were the smallest. However, the test set was very small, containing only 7 instances, of which only 2 were easy. This makes the numbers less representative of the actual expected performance when compared to the other ensembles. The number of PMP instances also had a major impact on the ensemble trained on PMP and SCP instances. Of 52 instances used for training, 50 were difficult and only 2 were easy. This resulted in an ensemble that did not have an adequate amount of training data about easy instances, a factor that helps explain the low test MF1 and MSAT $_{\mu}$ .

In addition to the performance on the small dataset, it was also verified how well the three ensembles perform on complete executions (big dataset). Table 15 shows the results for each of those ensembles separately and also includes other estimators for comparison. Starting with the case where the test set was composed of instances of the problem SCP, the CDF of the gamma distribution achieved the highest MSAT $_{\mu}$ , being followed by the RFR trained on the instance scp42. That RFR model being the second and surpassing the ensemble (the third place) is not surprising, because it was trained exclusively on complete executions of an instance from the SCP problem. However, it is important to mention that its training data was not considered during the model’s evaluation, similarly to what was done in Section 4.4.2. Given the fact that this approach leads to differences in the test

datasets used by the estimators, the results for that RFR model in that table cannot be reliably compared to those of the other estimators. As in the aforementioned section, these results were presented together for the sake of completeness and ease of comparison. However, removing the instances *scp42* and *pmed1* from all test datasets only leads to slight changes in the presented values.

In the cases where the test problem is the SCP or the QAP, no changes in the  $MSAT_\mu$  were observed within two decimal places (considering truncation). For that reason, the respective results in Table 15 would not change. However, when the test problem is the PMP, more relevant changes were observed when evaluating executions of all instances and those of the remaining easy instance (*pmed6*). Specifically, the  $MSAT_\mu$  for the estimators *Ensemble*, *CDF (gamma)*, *CDF (normal)*, *RFR (scp42)*, and *RFR (pmed1)* were, respectively, 0.62, 0.53, 0.33, 0.59 and 0.47 in the scenario “all instances”, and 0.46, 0.34, 0.23, 0.45 and 0.54 in the scenario “easy instances”. Those new values maintain the order of the top three estimators in each scenario.

Continuing with the analysis of the table’s results, and still in the case where the test problem is the SCP, the remaining two estimators performed significantly worse. In the case of the RFR trained on the executions of the instance *pmed1*, it still performed better than the CDF of the normal distribution, despite the fact that it was not trained on any difficult instance. It is also important to mention that, in the same section of the table, the scenarios “difficult instances” and “all instances” have the same results because the dataset used in the present work did not include any easy SCP instance.

Regarding the  $MSAT_\mu$  results for the case where the test problem is the PMP, the Ensemble appears to outperform all the other baseline methods when all instances in the dataset are considered and also when only the difficult ones are examined. If the test is conducted only on easy instances, that estimator falls to second, and the RFR trained on the instance *pmed1* takes the first position. However, while the ensemble was evaluated on executions of the instances *pmed1* and *pmed6*, the RFR model faced only the *pmed6*.

Switching the attention to the section of the table where the test executions belong to the QAP problem, no “\*” appears beside the RFR models, which indicates that no QAP instance was removed from the test set during the evaluation. The best  $MSAT_\mu$  is achieved by the CDF of the gamma distribution in all three variants of the test set. The ensemble, on the other hand, ranks as second only if all instances are considered. In the subset of difficult instances, it is the third, showing an  $MSAT_\mu$  slightly behind the one of the RFR trained on the instance *scp42* (the difference is not visible with two decimal places). In the case of the easy QAP instances, the ensemble is also the third, but not far behind the RFR trained on the *pmed1* instance. Regarding the results for the other metrics, the comments that could be made are the same as those made in the context of Table 11.

Given those results, their statistical significance can also be verified. The presence of 5 different estimators allows the use of the Friedman and the Nemenyi tests. However, because no cross-validation was performed, the multiple datasets that those methods require were represented by each execution. For example, for the evaluation of the statistical significance of the results for PMP instances, the metric was the  $MSAT_\mu$  and each of the more than 100 executions of that problem was considered a dataset. They are individual time series and have an associated MSAT. The use of those tests also required the evaluation of all estimators on the same data. For that reason, executions of the instances *scp42* and *pmed1* were not considered.

Starting with the Friedman test, the  $p$ -values for all tested cases were approximately 0, meaning that the null hypothesis can be rejected and that there is at least one estimator that significantly differs from the others in each case. The results of the Nemenyi test are provided by the CD diagrams in Figure 11. In those images, the names *Ensemble*, *CDF (gamma)*, *CDF (normal)*, *RFR (scp42)*, and *RFR (pmed1)* are replaced by *E*, *G*, *N*, *S*, and *P*, respectively. The values shown indicate that the ensemble, the CDF of the gamma distribution, and the RFR model trained on executions of the instance *scp42* are, in most cases, among the three with the best average ranks. The ensemble figures in the top three in all cases, showing the benefit of training on multiple instances of different difficulties. However, when it comes to the significance of the difference between estimators’ results, there is always another estimator whose estimates are not significantly different from the ensemble’s. In Figures 11a and 11f, the performance of the ensemble is significantly different from the estimator with the best average rank. In all other cases, the difference is not significant. It is also important to note that, due to the varying number of executions per problem and per difficulty, the value of the CD varied from case to case.

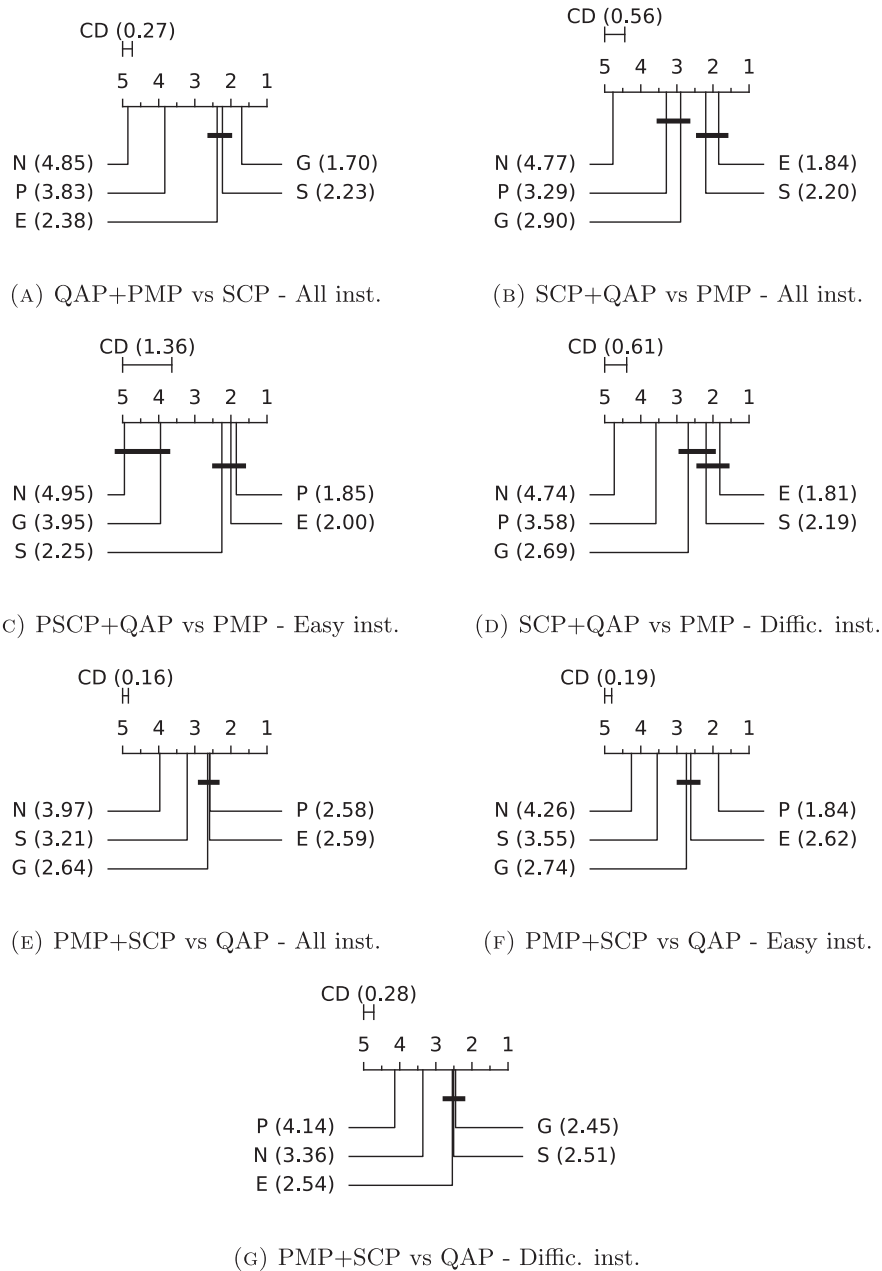


FIGURE 11. Critical Difference Diagrams showing the statistical significance of the results. Estimators connected by a horizontal bar are not significantly different in performance.

Finally, an additional piece of information that can be provided about the results is related to the hyperparameters used in each ensemble, which are shown in Table 16. From the values presented, it can be seen that the feature *iteration* was included in nearly all base models. In those cases, the feature importance was always 0.7 or higher, reaching 0.96 and 0.99 in the regressors trained on easy and difficult instances (respectively) of

TABLE 16. Hyperparameters of the models of the interproblem test.

Role	Hyperparameter	Ensembles		
		QAP+PMP	SCP+QAP	PMP+SCP
Classifier	training features	1, 2, 6, 7, 8, 9	3, 6, 7, 8, 9, 10	1, 2, 10
	n_neighbors	38	30	42
	weights	uniform	distance	distance
	$p$	euclidean	euclidean	manhattan
Regressor (easy instances)	training features	4, 6, 7, 9, 10	1, 2, 3	1, 9
	n_estimators	41	26	10
	max_depth	23	10	19
	min_samples_leaf	21	20	14
Regressor (difficult instances)	training features	1, 2, 3, 6, 7	1, 3, 4, 5	1, 8
	n_estimators	32	10	99
	max_depth	12	26	11
	min_samples_leaf	12	25	18

the ensemble PMP+SCP. For the two base models in that specific ensemble, the output was dominated by the number of the iteration, and the behavior of the curve of the estimates showed very little variation, regardless of the test execution being evaluated. That is one of the main factors behind the poor performance on easy QAP instances, because the learned behavior was mainly that of difficult instances (only two easy instances were used for training, as previously mentioned in this section).

#### 4.5. Discussion

The results presented in Sections 4.4.2 and 4.4.3 for complete executions show that the ensemble approach proposed in this work has a performance close to what is observed for other top estimators, sometimes surpassing their scores, and other times being not far behind. From the perspective of the  $MSAT_{\mu}$ , the ensemble also does not differ significantly from those in the first place in most cases and reaches the first position on three occasions: (1) difficult instances during the cross-validation; (2) difficult PMP instances in the interproblem evaluation; and (3) among the set of all PMP instances. However, this third occasion is a case where the representativeness of the results can be questioned because of the small number of easy PMP instances. Also in the interproblem evaluation, the ensemble ranked close to first among the difficult QAP instances. Given that situation, there is evidence that the method proposed in this work produces a stopping rule that is competitive when compared to other top performers, but mainly on difficult instances.

The CDF of the gamma distribution, on the other hand, showed a more consistent performance in the context of the cross-validation, which places it as the best estimator when it comes to the cross-validation score based on  $MSAT_{\mu}$ . Additionally, the interproblem evaluation clearly presents it as the best estimator in the case of SCP instances. However, this work did not test the possibility of combining that method with the CDF applied to a reflected cost distribution, where the latter would be used in situations where the original distribution is negatively skewed. That approach has the potential to improve the CDF's results, as it could treat positively and negatively skewed distributions with possibly more appropriate estimators. It would be similar to what the ensemble-based method does (a dynamic estimator selection), but dependent on the skewness of the cost value distribution, rather than on instance difficulty.

In the cases of the ensemble and of the CDF of the gamma distribution, the problems of stagnation and of temporary falls in the estimates were not treated. In the case of the CDF, they were the main ones responsible for the high MAE and RMSE observed in the executions of many instances, which were mostly easy instances. The use of those metrics was especially pertinent because they allowed the detection of those situations on some occasions, particularly the problem of stagnation of estimates, which can be easily recognized through

a high MAE. The proposed ensemble-based method expects a secondary termination criterion consisting of a maximum number of iterations, but it was not applied during the tests (although the size of the executions could be seen as a limit in the number of iterations). Despite that, all stopping rules can benefit from that secondary termination criterion, because it would always guarantee the stopping of the metaheuristic at some moment, even if stagnation occurs.

As for the RFR models trained on single instances, the average ranks shown in Figures 10 and 11 demonstrate that they perform better on instances of similar difficulty to that of the instance used for training, confirming a limitation that is mentioned in Appendix A. Despite that, when it comes to generalization capabilities, it can be said that the model trained on a difficult instance (scp42) is preferable over the one trained on an easy instance (pmed1), because even if it might take much longer than needed to stop an execution from an easy instance, it is often better than stopping too early in a difficult instance, in a moment when the incumbent solution is still of poor quality.

The CDF of the normal distribution was the estimator that performed the worst in all tested cases, but a poor result was already expected, given the limitations presented in Appendix A. The truncated normal might be able to improve the situation in some of the test executions if good lower bounds are available. However, if those values are available, a stopping rule based on the optimality gap could be used instead, as suggested by Resende and Ribeiro [55] for the Lagrangian GRASP. Given that situation, the applicability of the CDF of the standard normal distribution appears to be more limited than that of an RFR model trained on a single instance, because it depends on where the  $z$ -score of the optimal solution is located in the PDF of the distribution.

Regarding the architecture proposed for the ensemble, the combination of specialist models using a DRS approach was successful in creating a competitive method and reaching, in most cases, a better generalization across difficulties when compared to the models trained on a single instance. However, depending on the test dataset, the performance on easy instances was inferior to that of the CDF of the gamma distribution or to that of the RFR trained on executions of the instance pmed1. This is a situation that was caused by difficulty misclassification and by an underperforming regressor trained on easy instances. Nevertheless, in the latter case it might be related, at least in part, to the data used for the training.

For most easy instances, the optimal solution is reached before iteration 10 000, a fact that can be visualized in Figure 5. That same figure also shows that, in those cases, the number of minimum changes is usually smaller than 10. This means that, in most executions from easy instances, the details of the curve of remaining minima up to the optimal solution are described only by the few iterations where changes to the minimum occur (all the other 100 iterations describing the execution are just zero). On the other hand, the behavior of executions from difficult instances is represented by several times more iterations, because the last change in the minimum usually occurs after the iteration  $10^5$ , allowing the description of the curve of remaining minima to incorporate observations at every 10 000 iterations before that minimum is reached. However, this is a hypothesis that remains to be tested.

A concrete limitation observed in the approach taken to evaluate the ensemble resides in obtaining the training data. That happens because each execution took from hours to days to be generated. The training of the ensembles also presented a similar challenge, because the use of HPO resulted in training times longer than 24 h. The use of the sample dataset was intended to reduce the amount of resources and time needed to train the models. However, while it allowed the use of HPO, it still required the generation of the executions from which it was sampled.

Regarding the HPO procedure, model selection focused exclusively on the performance in the test set led to a curious situation in the case of the interproblem evaluation, because the test  $MSAT_{\mu}$  of one of the ensembles was much higher than the training  $MSAT_{\mu}$ . However, this kind of situation might be mitigated by including the performance on the training set in the fitness calculation in some way. The use of HPO also resulted in many ensembles with base models where the feature *iteration* achieved a very high importance. In some of those cases, that represented a problem, because the estimated curve of remaining minima showed little to no variation when complete executions were evaluated. The exploratory experiments tried to mitigate that situation by removing that feature or limiting its influence, but this resulted in a poorer fitness. The use of this feature also leaves

questions about the behavior of the estimates after iteration  $10^6$ , because this is the maximum iteration number available in the dataset.

Still in the context of the features employed in this work, the new approach to the calculation of remaining minima allowed the ensemble to learn behaviors that foresee a value greater than zero at the end of an execution. However, no comparison with the calculation method of Mattos [38] was provided, which means that it is not possible to discuss the impact of the change beyond what is already done in Appendix A.

Finally, the use of cross-validation, interproblem evaluation, and custom metrics showed that the ensemble-based approach proposed in this work was capable of generalizing even to unknown problems, which means that it might be applicable to standard GRASP implementations for multiple problems, depending on the data used for training and the type of instance being optimized. The  $MSAT_\mu$  was also effective in summarizing the performance across a large number of executions into a single value. However, it is important to mention that the use of a custom evaluation method (test cases, dataset and metrics) made it not possible to directly compare the results obtained in the present work to those obtained in previous works. Additionally, when it comes to comparisons with CDF estimates, it is important to note that the main target variable in the dataset is also an approximation. In other words, *remaining\_mins* was calculated based on sample probability, which means that, except for the iterations after a known optimal solution is found, the real probability of improvement is always unknown and may differ from the sample probability. Similarly, the CDF is the probability of finding a value smaller than or equal to the value being evaluated, and, for that reason, it is also an approximation.

## 5. CONCLUSION

### 5.1. Contributions and limitations

This work sought to analyze the limitations of existing stopping rules for the GRASP metaheuristic and propose an alternative ensemble-based rule that improves on some of them. Regarding the first objective, a series of shortcomings was described for a group of existing techniques. The comparison between rules indicated that the probabilistic approach based on the CDF of the gamma distribution was the most effective, although it faced problems in some easy optimization instances. It was also demonstrated that, in that method, the cost distribution does not need to be reflected if it has positive skewness.

As for the second objective, the proposed ensemble-based stopping rule was able to improve on the limitations of an already existing machine learning-based approach by working with a more diverse training dataset and introducing awareness about instance difficulty, which allowed it to be more flexible to instances of different difficulties. It produced better results than the CDF of the standard normal distribution, which performed poorly because normality is not guaranteed and the applicability of the CDF depends on how far from the mean the standardized cost of the optimal solution is. When compared to the CDF of the gamma distribution, the new rule showed a performance that indicates that it is competitive on harder instances. The ensemble also showed generalization capability on instances of problems that were not involved in its training. The tests performed included two custom similarity metrics, SAT and MSAT, which were proposed to measure how well the estimators performed at stopping a GRASP execution at expected moments. In the case of the evaluation of multiple executions at the same time, the average of the MSAT values was also effective at summarizing the performance into a single value. Compared to works that proposed previous stopping rules, the tests were performed on a much larger dataset, which, despite being imbalanced, covers more optimization problems or instances in most cases.

Regarding the limitations of this work, one of them is that the proposed stopping rule did not perform as well on easy instances as it did on difficult instances. However, this is not a big problem because some easy instances may be solved to optimality by exact methods. Another limitation is that the applicability of the method is restricted to the standard GRASP, and it is not possible to guarantee a satisfactory generalization on every instance of every problem. Its use in variants of the metaheuristic, if possible, may require modifications to the method. This work also did not solve the problem of requiring a large amount of training data, although the use of the sample dataset demonstrated that it is possible to use a smaller training dataset if the execution

data are representative of the characteristics of the complete executions from which they derive. As for analysis and comparisons, they did not include all existing stopping rules for GRASP.

## 5.2. Future work

The ensemble-based stopping rule showed promising results in difficult instances, but there is still room for improvement. The next steps that the authors aim to tackle are the improvement of the performance on easy instances and the possibility of using ensemble methods to combine estimators from existing approaches, like the CDF of different distributions and specialist machine learning models. Metalearning approaches where the ensemble itself creates specialist models could also be tested.

In addition to that, future works could try to change the MSAT metric to also account for the divergence between the estimates and the targets, which would remove the need for the metrics MAE and RMSE. The dataset could also be expanded to include execution data from more optimization problems. In that same context, synthetic executions could be used to increase the diversity of the training data without requiring the metaheuristic to be run. Finally, the applicability of the proposed stopping rule could be tested on variants of the GRASP metaheuristic and also on other metaheuristics. In the case of GRASP variants, this includes GRASP with path-relinking and Reactive GRASP.

## FUNDING

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

## DATA AVAILABILITY STATEMENT

The raw version (only cost values) of the dataset used in this work is available on Zenodo [41], under the reference <https://doi.org/10.5281/zenodo.15522802>.

The source code will not be made public, but the GRASP implementation for the PMP problem that was used in this work is available on GitHub [39], under the reference [https://github.com/GuilhermeCaeiro/grasp\\_pmedian\\_cpp](https://github.com/GuilhermeCaeiro/grasp_pmedian_cpp).

## REFERENCES

- [1] H. Alibrahim and S.A. Ludwig, Hyperparameter optimization: comparing genetic algorithm against grid search and bayesian optimization, in 2021 IEEE Congress on Evolutionary Computation (CEC) (2021) 1551–1559. DOI: [10.1109/CEC45853.2021.9504761](https://doi.org/10.1109/CEC45853.2021.9504761).
- [2] M. Alicastro, D. Ferone, P. Festa, S. Fugaro and T. Pastore, A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems. *Comput. Oper. Res.* **131** (2021) 105272.
- [3] A. Arbelaez and B. O’Sullivan, Learning a stopping criterion for local search, in Learning and Intelligent Optimization, edited by P. Festa, M. Sellmann and J. Vanschoren. Springer International Publishing, Cham (2016) 3–16.
- [4] J. Beasley, A note on solving large  $p$ -median problems. *Eur. J. Oper. Res.* **21** (1985) 270–273.
- [5] J. Beasley, An algorithm for set covering problem. *Eur. J. Oper. Res.* **31** (1987) 85–93.
- [6] J.E. Beasley, Or-library: distributing test problems by electronic mail. *J. Oper. Res. Soc.* **41** (1990) 1069–1072.
- [7] T.C. Belding, The distributed genetic algorithm revisited, in Proceedings of the 6th International Conference on Genetic Algorithms. San Francisco, Morgan Kaufmann Publishers Inc. CA, USA (1995) 114–121. DOI: [10.48550/arXiv.adap-org/9504007](https://doi.org/10.48550/arXiv.adap-org/9504007).
- [8] C.G.E. Boender and A.H.G. RinnooyKan, Bayesian stopping rules for multistart global optimization methods. *Math. Program.* **37** (1987) 59–80.
- [9] W. Bożejko, A. Burduk, K. Musia and J. Pempera, Neuro-tabu search approach to scheduling in automotive manufacturing. *Neurocomputing* **452** (2021) 435–442.
- [10] L. Breiman, Random forests. *Mach. Learn.* **45** (2001) 5–32.
- [11] R.E. Burkard, S.E. Karisch and F. Rendl, QAPLIB – a quadratic assignment problem library. *J. Glob. Optim.* **10** (1997) 391–403.

- [12] J.T.H.A. Cabral, R.A. Araujo, J.P. Nobrega and A.L. deOliveira, Heterogeneous ensemble dynamic selection for software development effort estimation, in 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI) (2017) 210–217. DOI: [10.1109/ICTAI.2017.00042](https://doi.org/10.1109/ICTAI.2017.00042).
- [13] K. Carling and M. Han, GRASP and statistical bounds for heuristic solutions to combinatorial problems. Technical report, Dalarna University (2016). <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A946358&dswid=-5516>.
- [14] K. Carling and X. Meng, Confidence in heuristic solutions? *J. Glob. Optim.* **63** (2015) 381–399.
- [15] H.D.P. Carvalho, J.F.L. deOliveira and R.A.A. Fagundes, Dynamic selection of ensemble-based regression models: systematic literature review. *Expert Sys. Appl.* **290** (2025) 128429.
- [16] A. Corominas, On deciding when to stop metaheuristics: properties, rules and termination conditions. *Oper. Res. Perspect.* **10** (2023) 100283.
- [17] J. Demšar, Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7** (2006) 1–30.
- [18] A.M. Fathollahi-Fard, M. Hajiaghahi-Keshteli and R. Tavakkoli-Moghaddam, Red deer algorithm (RDA): a new nature-inspired meta-heuristic. *Soft Comput.* **24** (2020) 14637–14665.
- [19] G. Felici, D. Ferone, P. Festa, A. Napolitano and T. Pastore, A GRASP for the minimum cost SAT problem, in edited by R. Battiti, D.E. Kvasov and Y.D. Sergeyev. International Conference on Learning and Intelligent Optimization. Springer International Publishing, Cham (2017) 64–78.
- [20] T.A. Feo and M.G.C. Resende, Greedy randomized adaptive search procedures. *J. Glob. Optim.* **6** (1995) 109–133.
- [21] D. Ferone, P. Festa and F. Guerriero, The rainbow steiner tree problem. *Comput. Oper. Res.* **139** (2022) 105621.
- [22] E. Fink and K.B. Pratt, Indexing of Compressed Time Series. World Scientific (2004) 43–65.
- [23] S.N. Ghoreishi, A. Clausen and B.N. Joergensen, Termination criteria in evolutionary algorithms: a survey, in Proceedings of the 9th International Joint Conference on Computational Intelligence (IJCCI 2017) – IJCCI, INSTICC. SciTePress (2017) 373–384. DOI: [10.5220/0006577903730384](https://doi.org/10.5220/0006577903730384).
- [24] A. Goli, Integration of blockchain-enabled closed-loop supply chain and robust product portfolio design. *Comput. Ind. Eng.* **179** (2023) 109211.
- [25] A. Goli, Efficient optimization of robust project scheduling for industry 4.0: a hybrid approach based on machine learning and meta-heuristic algorithms. *Int. J. Prod. Econ.* **278** (2024) 109427.
- [26] A. Goli and E.B. Tirkolaee, Designing a portfolio-based closed-loop supply chain network for dairy products with a financial approach: accelerated benders decomposition algorithm. *Comput. Oper. Res.* **155** (2023) 106244.
- [27] A. Goli, A. Ala and M. Hajiaghahi-Keshteli, Efficient multi-objective meta-heuristic algorithms for energy-aware non-permutation flow-shop scheduling problem. *Expert Sys. Appl.* **213** (2023) 119077.
- [28] A. Goli, A. Ala and S. Mirjalili, A robust possibilistic programming framework for designing an organ transplant supply chain under uncertainty. *Ann. Oper. Res.* **328** (2023) 493–530.
- [29] C.R. Harris, K.J. Millman, S.J. vander Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J. Fernándezdel Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T.E. Oliphant, Array programming with NumPy. *Nature* **585** (2020) 357–362.
- [30] S. Herbold, Autorank: a python package for automated ranking of classifiers. *J. Open Source Softw.* **5** (2020) 2173.
- [31] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. The MIT Press (1992).
- [32] J.D. Hunter, Matplotlib: a 2D graphics environment. *Comput. Sci. Eng.* **9** (2007) 90–95.
- [33] D. Karaboga, B. Akay and N. Karaboga, A survey on the studies employing machine learning (ML) for enhancing artificial bee colony (ABC) optimization algorithm. *Cogent Eng.* **7** (2020) 1855741.
- [34] W. Li, G.-G. Wang and A.H. Gandomi, A survey of learning-based intelligent optimization algorithms. *Arch. Comput. Methods Eng.* **28** (2021) 3781–3799.
- [35] S. Liu, H. Wang, W. Peng and W. Yao, Surrogate-assisted evolutionary algorithms for expensive combinatorial optimization: a survey. *Complex Intell. Syst.* **10** (2024) 5933–5949.
- [36] S. Lloyd, Least squares quantization in PCM. *IEEE Trans. Inf. Theory* **28** (1982) 129–137.
- [37] G. Luque and E. Alba, Parallel Genetic Algorithms: Theory and Real World Applications. Studies in Computational Intelligence. Springer, Berlin, Germany, 2011 edition (2011). DOI: [10.1007/978-3-642-22084-5](https://doi.org/10.1007/978-3-642-22084-5).
- [38] G.C. Mattos, *Machine learning-based probabilistic stopping rule for the GRASP metaheuristic*. Master’s dissertation, Federal University of Rio de Janeiro, Rio de Janeiro, RJ (2021). <https://www.cos.ufrj.br/index.php/pt-BR/publicacoes-pesquisa/details/15/3022>.

- [39] G.C. Mattos, GRASP C++ code for the  $p$ -median problem for “Ensemble machine learning-based stopping rule for greedy randomized adaptive search procedure” (2026). [https://github.com/GuilhermeCaeiro/grasp\\_pmedian\\_cpp](https://github.com/GuilhermeCaeiro/grasp_pmedian_cpp).
- [40] G.C. Mattos, F.M.G. França, L.G. Simonetti and P.M.V. Lima, AIISR – AI inspired stopping rule for GRASP metaheuristic, in Anais do Simpósio Brasileiro de Pesquisa Operacional. Vol. 53. João Pessoa – Paraíba, Brasil, Galoa (2021). DOI: [10.59254/sbpo-2021-131575](https://doi.org/10.59254/sbpo-2021-131575).
- [41] G.C. Mattos, L.A.D. LusquinoFilho, L.G. Simonetti and P.M.V. Lima, Raw dataset (v1.0) for “Ensemble machine learning-based stopping rule for greedy randomized adaptive search procedure” (2026). <https://doi.org/10.5281/zenodo.15522802>.
- [42] A. Morales-Hernández, I. VanNieuwenhuysse and S. RojasGonzalez, A survey on multi-objective hyperparameter optimization algorithms for machine learning. *Artif. Intell. Rev.* **56** (2023) 8043–8093.
- [43] T.J.M. Moura, G.D.C. Cavalcanti and L.S. Oliveira, Evaluating competence measures for dynamic regressor selection, in 2019 International Joint Conference on Neural Networks (IJCNN). IEEE (2019) 1–8. DOI: [10.1109/IJCNN.2019.8851835](https://doi.org/10.1109/IJCNN.2019.8851835).
- [44] L.A. Neves, *Critérios de parada baseados em probabilidade bayesiana aplicados a heurísticas GRASP: um estudo experimental*. Master’s thesis, Federal University of the Stat of Rio de Janeiro, Rio de Janeiro, RJ, May (2009). <http://www.repositorio-bc.unirio.br:8080/xmlui/handle/unirio/12807>.
- [45] L.A. Neves, A.C.F. Alvim and M.G.C. Resende, Implementação e teste de critérios de parada para heurísticas GRASP, in Anais Do XLI Simposio Brasileiro De Pesquisa Operacional (SBPO). Porto Seguro (2009) 1989–1999. <https://www.researchgate.net/publication/255654512>.
- [46] V.D. Noghin, A combined approach to reducing the pareto set using linear or multiplicative scalarization. *Sci. Tech. Inf. Process.* **44** (2017) 373–378.
- [47] C.A.S. Oliveira, P.M. Pardalos and M.G.C. Resende, GRASP with path-relinking for the quadratic assignment problem, in Experimental and Efficient Algorithms, edited by C.C. Ribeiro and S.L. Martins. Springer Berlin Heidelberg, Berlin, Heidelberg (2004) 356–368.
- [48] C. Orsenigo and C. Vercellis, Bayesian stopping rules for greedy randomized procedures. *J. Glob. Optim.* **36** (2006) 365–377.
- [49] T. Pastore, C. Menna and D. Asprone, Bézier-based biased random-key genetic algorithm to address printability constraints in the topology optimization of concrete structures. *Struct. Multidiscipl. Optim.* **65** (2022) 64.
- [50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12** (2011) 2825–2830. <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>.
- [51] L.S. Pessoa, M.G.C. Resende and C.C. Ribeiro, Experiments with LAGRASP heuristic for set  $k$ -covering. *Optim. Lett.* **5** (2011) 407–419.
- [52] L.S. Pessoa, M.G. Resende and C.C. Ribeiro, A hybrid Lagrangean heuristic with GRASP and path-relinking for set  $k$ -covering. *Comput. Oper. Res.* **40** (2013) 3132–3146.
- [53] M. Prais and C.C. Ribeiro, Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS J. Comput.* **12** (2000) 164–176.
- [54] A. Ramdas, N.G. Trillos and M. Cuturi, On Wasserstein two-sample testing and related families of nonparametric tests. *Entropy* **19** (2017) 47.
- [55] M.G. Resende and C.C. Ribeiro, Optimization by GRASP: Greedy Randomized Adaptive Search Procedures. Springer New York, New York, NY (2016). DOI: [10.1007/978-1-4939-6530-4](https://doi.org/10.1007/978-1-4939-6530-4).
- [56] M.G. Resendel and C.C. Ribeiro, GRASP with Path-ReLinking: Recent Advances and Applications. Springer US, Boston, MA (2005) 29–63.
- [57] C.C. Ribeiro, I. Rosseti and R.C. Souza, Probabilistic stopping rules for GRASP heuristics and extensions. *Int. Trans. Oper. Res.* **20** (2013) 301–323.
- [58] S. Skipper and P. Josef, Statsmodels: econometric and statistical modeling with python. *SciPy* **7** (2010) 92–96.
- [59] P.N. Smyrlis, D.C. Tsouros and M.G. Tsipouras, Constrained  $k$ -means classification. *Eng. Technol. Appl. Sci. Res.* **8** (2018) 3203–3208.
- [60] M. Sulaman, M. Golabi, M. Essaid, M. Brévilliers, J. Lepagnet and L. Idoumghar, Random forest assisted differential evolution for multi-server congested  $p$ -median problem, in 2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI) (2023) 404–409. DOI: [10.1109/ICTAI59109.2023.00065](https://doi.org/10.1109/ICTAI59109.2023.00065).
- [61] Y. Sun, S. Wang, Y. Shen, X. Li, A.T. Ernst and M. Kirley, Boosting ant colony optimization via solution prediction and machine learning. *Comput. Oper. Res.* **143** (2022) 105769.

- [62] F. Tao, Y. Laili and L. Zhang, Brief History and Overview of Intelligent Optimization Algorithms. Springer International Publishing, Cham (2015) 3–33.
- [63] K. Taunk, S. De, S. Verma and A. Swetapadma, A brief review of nearest neighbor algorithm for learning and classification, in 2019 International Conference on Intelligent Computing and Control Systems (ICCS) (2019) 1255–1260. DOI: [10.1109/ICCS45141.2019.9065747](https://doi.org/10.1109/ICCS45141.2019.9065747).
- [64] The pandas development team. Pandas (2024) <https://doi.org/10.5281/zenodo.13819579>.
- [65] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, Í. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt and SciPy 1.0 Contributors, SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat. Methods* **17** (2020) 261–272.
- [66] A.S. Wicaksono and A. Aff, Hyper parameter optimization using genetic algorithm on machine learning methods for online news popularity prediction. *Int. J. Adv. Comput. Sci. Appl.* **9** (2018). DOI: [10.14569/IJACSA.2018.091238](https://doi.org/10.14569/IJACSA.2018.091238).
- [67] S.C. Yusta, Different metaheuristic strategies to solve the feature selection problem. *Pattern Recognit. Lett.* **30** (2009) 525–534.
- [68] M. Žižović, M. Žižović, N. Damljanović and K. Pavlović, Multiplicative method based on expected criteria values. *Rep. Mech. Eng.* **4** (2023) 317–325.

**Please help to maintain this journal in open access!**



This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting [subscribers@edpsciences.org](mailto:subscribers@edpsciences.org).

More information, including a list of supporters and financial transparency reports, is available at <https://edpsciences.org/en/subscribe-to-open-s2o>.

APPENDIX A. LIMITATIONS OF SOME EXISTING STOPPING RULES

In order to start this appendix, it is important to first establish a definition for instance difficulty that is pertinent to the present work. In that definition, an instance is said to be *easy* if, among the executions available for it in the dataset (10, 20 or 30), the corresponding standard GRASP implementation was able to achieve the best known solution in the literature, which might be optimal or not (the majority is), on all occasions. In cases where it did not, the instance was considered *difficult*. In other words, an instance is treated as *easy* if the average gap was 0 and as *difficult* otherwise. That difference is important because it has implications on how the curve of expected remaining minima behaves and also on the shape of the distribution of cost values.

Regarding the curve of expected remaining minima, it is the result of the counting of improved solutions observed in an interval of future iterations (similarly to what is described in Sect. 2.2.4). In easy instances, where the optimal solution is commonly found in early iterations, that curve also reaches zero as soon as the optimal solution is found, and, for that reason, it is usually narrow. On the other hand, in difficult instances, improvements to the solution are expected to happen through the entirety of the execution. Consequently, the number of remaining minima falls gradually until it reaches zero or a small number later in the execution, resulting in a wide curve.

In Mattos [38], the use of models trained on executions from a single instance is proposed, and, for that reason, instance difficulty is not considered. That approach figures as a limitation because those models are expected to perform better on instances whose difficulty is similar to that of the instance used for training. Specifically, if a model is trained on an easy instance, using the feature set independent of cost value that was proposed by that work, and is applied to a difficult instance, the estimates might reach a very small number of remaining minima or zero in early iterations.

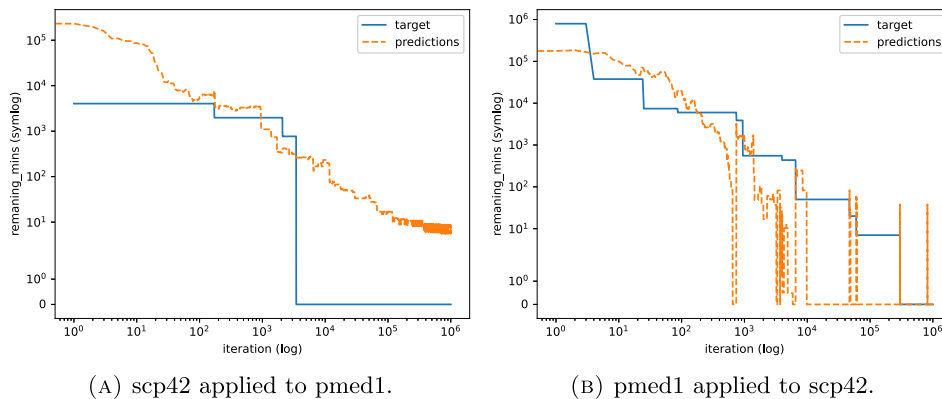


FIGURE A.1. Example of models trained on single instances being applied to instances of different difficulty.

Likewise, if the model is trained on a difficult instance and applied to an easy one, it might not recognize that an optimal solution was found and only estimate zero or a very small value later on in the execution.

The training features play a major role in that problem. In the case where cost value is not included in some way, the feature set was composed of the number of minimum changes, the number of the iteration, the number of iterations since the last minimum change, and the number of iterations since the current minimum appeared for the last time (it can appear multiple times). While that combination might allow for the identification of patterns, tests where iteration number was included were prone to result in models where that feature achieved very high importance, meaning that it ended up dominating the decision on whether to stop the metaheuristic.

In order to exemplify the entire situation, in Figure A.1, the results for two models are presented. They are random forest regressors, trained with hyperparameters used by Mattos [38]. Specifically, there were 10 trees in each model, and each tree had a maximum depth of 10 and at least 20 observations in each leaf. In Figure A.1a, the model trained on instance scp42, which is a difficult one, was applied to an execution belonging to instance pmed1, which is easy (information about instances is available in Sect. 4.1). Meanwhile, in Figure A.1b, the roles are inverted. The model was trained on instance pmed1 and the execution was from instance scp42. In those results, it is possible to see the previously mentioned problem of applying models of one difficulty to executions of another. Moreover, if the feature importances are inspected, iteration number achieved approximately 0.76 and 0.83 for the scp42 and the pmed1 models, respectively, which means that it had a very high impact on the decision process.

It is important to mention that the number of remaining minima was calculated according to the method described in Section 4.1. That information brings attention to another possible limitation, which lies in the approach taken to perform that calculation in Mattos [38], because the ever-shrinking counting window that it employs will cause the number of remaining minima to reach 0 at some point in an execution, making it not possible for the models to get an adequate insight into the real behavior of that variable near the end of an execution.

In addition to those points already presented, another restriction lies in the way in which the features are normalized. Iteration number reaches the upper bound used for normalization at iteration  $10^6$ , which means that it stops providing updated information past that point, despite the fact that difficult instances will generally require more than that number of iterations to reach optimality. In the case of cost value, it was normalized using the cost of the best known solution from the literature as a lower bound when used as a feature. However, for unknown instances, the best cost is expected to be unknown, which means that the results provided in these cases can only be interpreted as best-case scenarios.

One final drawback of the machine learning-based stopping rule is the time required to obtain the data used for training. Even if only one instance is used, many executions of size  $10^6$  will be required, as well as several hours or several days to generate them, depending on the instance used. It is a one-time procedure, but it is costly.

Moving to the relation that instance difficulty has with the shape of the distribution, Table A.1 shows that, for a set consisting of 180 (out of 184) instances present in the dataset of Section 4.1, the absolute difference between 0 and the observed skewness and excess kurtosis was always smaller than 1 in difficult instances, while in easy instances it varied considerably. Additionally, after visual inspection, it is possible to see that, in some easy instances, the distribution does

TABLE A.1. Limited summary of some statistics of the dataset employed in Section 4.1. Values are based on the averages of the statistics observed for executions of each instance.

Statistic	Easy	Difficult
Min Skewness	-29.14	-0.25
Max Skewness	6.54	0.77
Min Kurtosis	-1.48	-0.55
Max Kurtosis	1389.49	0.90
%  Skewness  > 1	25.27%	0.00%
%  Kurtosis  > 1	31.86%	0.00%
Min $Z$ -score LB	-47.73	-85.92
Max $Z$ -score LB	-0.16	-2.51

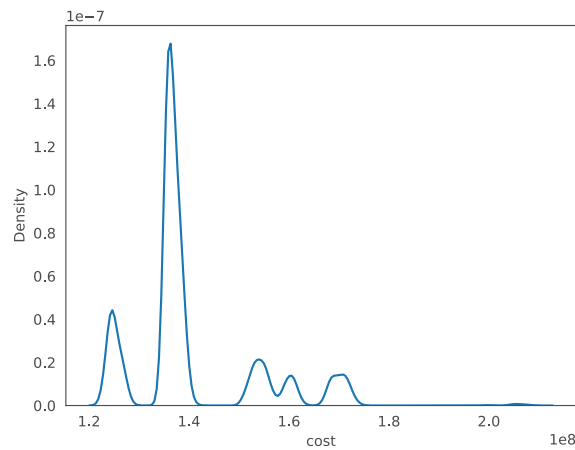


FIGURE A.2. Cost value distribution of the instance tai20b.

not even resemble a normal distribution, as in the case of instance tai20b, which looks like a sequence of bell curves, as shown by Figure A.2.

The observation that the cost value distribution can differ from a normal distribution by being significantly skewed, kurtotic or by having an irregular shape conflicts with the assumption made by Ribeiro *et al.* [57] that the distribution is approximately normal. Moreover, if the CDF of the non-truncated standard normal is used, a probability smaller than  $10^{-6}$ , which would be equivalent to 0 remaining minima in the test method employed by that same work, would require a minimum cost with a  $z$ -score smaller than  $-4.75$  (value obtained using the Percentage Point Function of the normal distribution). That is a value that, according to Table A.1, some instances would not be able to reach, regardless of being easy or difficult, because their lower bounds (best known solutions) return a higher value. A consequence of that is a stagnation of the estimates in cases where that  $z$ -score cannot be reached, meaning that, even if the optimal solution is evaluated, a high probability could be returned. As for how high it can be, even something in the order of  $10^{-1}$  could be expected, like in the case of the instance tai64c (an easy one), whose optimal cost returns a probability of approximately 0.18, showing that, for its case, the CDF of the non-truncated distribution would not be reliable.

An alternative to that situation, which is already proposed by the authors of the technique, is the use of the CDF of the truncated normal. However, that approach stumbles on a major problem, which is the difficulty in obtaining quality lower bounds for the cost value. Quality is relevant because, depending on how far the bound is from the optimal cost, the probability might reach zero too early or continue to face stagnation. In the first case, it happens when the lower bound is much higher than the optimal cost and might be reached with ease, while the second case occurs if the bound is smaller, meaning that a probability of 0 will never be reached and the estimate might stagnate at an undesirably high value. An example of that second problem is the instance tai35b, where a lower bound 0.01% smaller than the optimal,

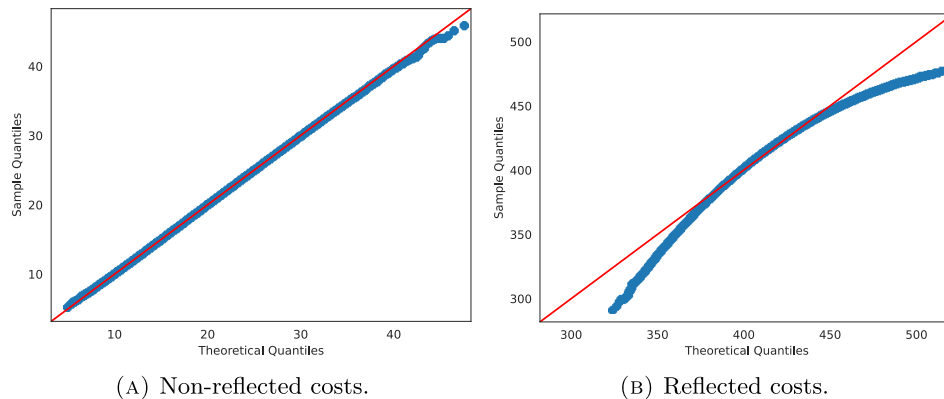


FIGURE A.3. Q-Q plots comparing the distribution of the instance scp42 and the gamma distribution.

coupled with an upper bound calculated as suggested by Ribeiro *et al.* [57], results in a probability of approximately 0.03 when the optimal solution is evaluated.

Among the stopping rules proposed for GRASP, the approach based on the gap between statistical bounds [13], briefly mentioned in Section 2, in addition to being able to stop the metaheuristic on a quality basis, could also provide a way to obtain the lower bound for the truncated normal and for the machine learning-based approach. However, according to the results that its authors provide, the method did not guarantee the tightness of the bounds, did not guarantee that the optimal solution would be covered, and, depending on the instance, it may also require a large amount of data.

Regarding the first and the second limitations, among the two proposed estimators, the Weibull point estimator managed to achieve tighter bounds, but presented a low coverage rate in most of the instances used for evaluation. On the other hand, the second order Jackknifing point estimator had wider bounds and a higher coverage rate. As for the amount of data (cost values) that the method requires, it is not clear how many iterations each of the suggested 100 replicates (executions) of the metaheuristic had, because the work mentions numbers of local searches (which could mean iterations or local search steps in each local search phase). Nevertheless, if the highest running times in Carling and Meng [14], which are used as reference for instances of some problems, are taken into consideration, each replicate could have, at least, tens of thousands of iterations.

One way to avoid dealing with truncation is to use the probabilistic stopping rule based on the gamma distribution, which was described in Section 2.2.3. However, there are two problems with this approach. The first has to do with the necessity of the reflection procedure, because its authors argue that, in minimization problems, a longer tail to the left (negative skewness) is to be expected, a characteristic that they verified in their instances for the MinCostSAT problem. Despite that, in the present work, the majority of the instances (also from minimization processes) showed positive skewness, putting into question the need for a reflection procedure in those cases.

In order to verify that, in Figure A.3, Q-Q plots for an execution of the instance scp42 are provided. That execution has skewness and kurtosis of 0.44 and 0.26, respectively. In Figure A.3a, the non-reflected distribution is shown to match well the theoretical gamma distribution, while Figure A.3b shows that the same does not apply if reflection is used.

Based on those results, it is possible to argue that, for positively skewed distributions, reflection might have the adverse effect of providing worse results, an argument that can be strengthened by evaluating the CDF at the best cost from the literature (references about those costs are mentioned in Sect. 4.1). In the case of the non-reflected distribution, the probability returned is approximately  $8.6 \cdot 10^{-10}$ , while the estimate gets stagnated at approximately  $3 \cdot 10^{-4}$  if reflection is used. A similar situation was seen in many other instances whose distribution was positively skewed. On the other hand, in the case of instances that show negative skewness, the present work was unable to verify the suitability of the reflection procedure because less than 10% of the instances in the dataset shared that characteristic, and their distributions often presented an irregular shape.

In order to gather further information on which distribution is more appropriate to be used, the Wasserstein Distance [54] was used to evaluate the divergence between the cost distribution and normal and gamma distributions fitted to the parameters obtained using MLE. That metric calculates the minimal effort necessary to transform one distribution into another and is applied, in the present case, to check which alternative is the closest to the empirical cost distribution (in

TABLE A.2. Number of instances where each fitted distribution presented the smallest Wasserstein Distance.

Skewness	Difficulty	Normal	Gamma (non-ref. costs)	Gamma (ref. costs)
Negative	Easy	4	2	7
	Difficult	0	0	1
Positive	Easy	19	59	2
	Difficult	0	87	1

TABLE A.3. Instances per probability range resulting from the use of the CDF of the gamma distribution.

Probability range	Median probability	Instances		
		Easy	Difficult	Total
$\geq 10^{-6}$	$1.1 \times 10^{-3}$	74	10	84
$< 10^{-6}$	$3.0 \times 10^{-14}$	19	79	98

the case of the gamma distribution, the use of reflection was also tested). In the test, 182 instances were considered and each was represented by one execution.

The results of that comparison are presented in Table A.2 and show that, in general, the gamma distribution fitted on non-reflected costs is the best option, presenting the smallest distance in nearly all 89 difficult instances considered and also in the majority of the 93 easy instances. As for the use of the gamma distribution fitted to reflected costs, it had the smallest distance in only 11 instances, and, among them, 8 had negative skewness (out of 14 executions with that characteristic among the 182 executions). Considering that the use of reflection led to the smallest distance in the majority of the instances with negative skewness, these results point to the possibility that it might be appropriate to use non-reflected cost for instances with positive skewness and reflected costs for cases with negative skewness. However, properly testing that possibility would require a larger number of instances that produce a negatively skewed distribution.

Finally, with respect to the second limitation of the use of the gamma distribution, it lies in the observation that, although the CDF performs well on difficult instances (when reflection is not used), its estimates often stagnate at some probability above  $10^{-6}$  on easy instances. This problem is tied to the previous verification that, on multiple occasions, the gamma distribution did not fit easy instances well. In Table A.3, the CDF at the best known solution from the literature was calculated for one execution of each instance, and the instances were grouped based on whether the resulting probability was below  $10^{-6}$  or not. It is possible to see that not only the vast majority of the easy fell in the group with CDF equal or greater than that threshold, but also that, in that group, stagnation occurred at approximately  $10^{-3}$  or more in at least half of the instances (observation based on the median probability).