IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 5, 2021, accepted August 9, 2021, date of publication August 23, 2021, date of current version August 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3106815

Sequence-Based Selection Hyper-Heuristic Model via MAP-Elites

MELISSA SÁNCHEZ, JORGE M. CRUZ-DUARTE, (Member, IEEE), JOSÉ C. ORTIZ-BAYLISS, (Member, IEEE), AND IVAN AMAYA, (Member, IEEE)

School of Engineering and Sciences, Tecnologico de Monterrey, Monterrey, Nuevo Leon 64849, Mexico

Corresponding author: Ivan Amaya (iamaya2@tec.mx)

This work was supported in part by the Consejo Nacional de Ciencia y Tecnología (CONACYT) Basic Science Project under Grant 287479, and in part by the Research Group with Strategic Focus in Intelligent Systems at Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM).

ABSTRACT Although the number of solutions in combinatorial optimization problems (COPs) is finite, some problems grow exponentially and render exact approaches unfeasible. So, approximate methods, such as heuristics, are customary. Each heuristic usually specializes in specific kinds of problems. Hence, other approaches seek to merge their strengths. One of them is selection hyper-heuristics. However, they usually provide scarce information about their sensitivity. Illumination algorithms may fix this issue since they focus on exploration rather than exploitation while preserving the best solutions under different criteria. Still, literature falls short when merging both approaches, representing a knowledge gap. This work tests the feasibility of using an illumination algorithm, MAP-Elites (ME), for tuning a sequence-based selection hyper-heuristic model for Balanced Partition problems. We choose ME since other researchers have successfully applied it to a different COP. So, we may achieve a hyper-heuristic that represents the best combination of heuristics while simultaneously gaining intel on the performance of diverse alternatives. Our approach operates by creating a multi-dimensional map, where each design variable represents the application of a heuristic. Afterward, ME generates mutated sequences and tests them to determine if they represent a better-performing solution. We consider 1500 instances that include easy and hard instances, analyzed under different scenarios to test our approach. We also include limit instances that are neither easy nor hard. Our resulting data support the proposed approach, as it performs toe-to-toe with a synthetic oracle and may even outperform it. This represents an outstanding result, since a brute-force approach is needed to achieve such an oracle. So, merging ME and hyper-heuristics is a path worth pursuing. We also present how each parameter affects the model performance and identify the critical and virtually irrelevant ones. This serves as the groundwork for future works that focus on exploiting the most relevant parameters.

INDEX TERMS MAP-elites, balanced partition, hyper-heuristic, combinatorial optimization, heuristic.

I. INTRODUCTION

We can use a Combinatorial Optimization Problem (COP) to represent real-life problems. Here, values must be selected from a discrete domain while aiming to reach an optimal goal. Examples include Balanced Partition and Job-Shop Scheduling. The first case is akin to situations with multiple operators, e.g., multi-core problems [1]. The second one usually maps the planning of manufacturing processes [2]. Of course, literature is prolific with studies about other COPs [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Kai Li.

Heuristics are common, straightforward methods for solving real-world COPs [4]. Despite their success, they are problem-specific, and their mileage may vary as they do not guarantee optimality. Their adequate performance is often restricted to instances of the same nature as those the heuristic was built for. So, performance levels for other kinds of instances remain unclear and one must test them. These drawbacks stem from their likeness to simple *rules-of-thumb*. Hence, researchers have sought alternatives for improving their trustworthiness [5]–[7]. Nonetheless, we bring up only two of them: metaheuristics and hyper-heuristics.

Metaheuristics (MHs) have existed for quite some time. It is customary for them to be linked to a biological metaphor.

Genetic Algorithms (GAs), based on the theory of evolution [8], and Simulated Annealing (SA), inspired in crystallization phenomena during the annealing of metals [9], are among the earliest approaches. Since they are rather straightforward to implement, MHs have become mainstream for solving optimization problems, including COPs [10]–[13]. Nowadays, research efforts include the automatic generation of metaheuristics tailored to a given set of problems [14].

Similarly, hyper-heuristics (HHs) refers to a reasonably recent proposal that has drawn attention because of its performance [15], [16]. There are several types of HHs, and their classification has grown from a relatively simple approach to a complex mixture of perspectives [16]. Amongst them reside selection hyper-heuristics, where the solver selects a low-level strategy (such as a heuristic) from a pool (e.g., the set of available heuristics) at each step of the solution. Selection hyper-heuristics allow taking advantage of the strengths of each available solver, as each one only needs to solve a portion of the problem.

However, selection hyper-heuristics exhibit a couple of drawbacks. One of them is the need to address each problem instance and identify the current state of its solution so that one may select a proper heuristic. The other one is that the selection model must be able to traverse a search space that can grow drastically as we add more heuristics. The reason is that a selection hyper-heuristic must choose a heuristic at each step of the solution process. So, an exhaustive approach implies evaluating the effect of all heuristics every time the system must decide which heuristic to apply. This, of course, grows exponentially w.r.t. the number of available heuristics. Although this approach sounds similar to algorithm portfolios, they perform a single selection per problem instance (when the search starts). Instead, HHs execute several selections throughout the search, as we just mentioned.

For the sake of clarity, we now mention a brief example. Let us assume that algorithm portfolios can be laid out as a sequence of the decisions they carry out for solving a given problem instance. From this perspective, they have a single element in the sequence. Conversely, hyper-heuristics may exhibit as many as desired. So, one may consider selection hyper-heuristics a superset of algorithm portfolios.

A HH does not directly solve a problem instance. Instead, it seeks the right approach for a given situation. Then, HHs operate on the solver space and not on the problem space [17]–[19]. This approach has proved its worth when solving COPs. For example, Garza-Santisteban et al. used a Simulated Annealing (SA) algorithm to deal with JSSPs, managing to outperform the oracle in some scenarios [20]. Yu et al. developed a hyper-heuristic resulting from the hybridization of a multi-decoding framework with an Evolutionary Algorithm [21]. Wu et al. developed several HHs based on SA and using five heuristics that operate as their foundations [22]. There are way more exciting works, but it is unfeasible to discuss them all here. So, the interested reader is directed towards [3], [15], [16], [18] for more information.

Illumination algorithms are another strategy for solving optimization problems that render an overview of the landscape associated with the fitness of the objective function. In doing so, they provide optimal solutions under different design considerations. The Multi-dimensional Archive of Phenotypic Elites (MAP-Elites, or simply ME) is an example of this kind of strategy [23], [24]. MAP-Elites explores the search space with a grid of high-performing solutions (*elites*) in terms of user-defined features.

Although illuminating a domain has some advantages, it also exhibits some drawbacks. The process is more widespread and thus it scans more of the search space. But, in doing so, ME analyzes promising solutions in less detail than with traditional optimization methods. Despite this, identifying promising solutions under different criteria may prove worthwhile. So, advantages stem from the insight of how solution combinations correlate with performance and on the finding of multiple potential solutions [23].

Illumination seems akin to multi-objective optimization [25]–[27]. But, the latter focuses on finding a set of best-performing solutions, based on the criteria from two or more objective functions. Conversely, illumination tackles a single objective function, although also provides a set of best-performing solutions. Illumination provides best-performing solutions for different regions of each design variable.

Up to now, researchers have restricted most applications of ME to robotic tasks, such as the evolution of robot arms [28], robot morphologies [28], and soft-bodied robots [23]. Besides, they have also applied ME to constrained optimization [24], [29]. To the best of our knowledge, literature contains no works where HHs meet ME. So, we aim to fill that knowledge gap by evaluating the feasibility of using ME for training selection hyper-heuristics that solve the Balanced Partition problem. The reason: traditional HHs fall short when providing information about their sensitivity, which can be alleviated by using an illumination algorithm throughout the training process. Since the problem can be coded as a sequence of decisions (i.e., heuristics), ME can be used for illuminating the search space of all possible combinations of heuristics. In this way, we may not only achieve a good solver, but also information about the way performance changes if the model, or its parameters, are altered. Hence, this work has three major contributions:

1) It proposes a sequence-based selection hyper-heuristic model powered by MAP-Elites and for solving the Balanced Partition problem;
2) It analyzes the effect of several parameters over the performance of the model, identifying those that are critical and those with virtually no effect; and
3) It demonstrates that the proposed model outperforms low-level heuristics, randomly-selected sequences, and even a synthetic oracle (under the right conditions).

We have structured this paper as follows. Section II summarizes fundamental concepts from our work and discusses recent works on selection hyper-heuristics and MAP-Elites. Then, Section III describes the proposed hyper-heuristic

model, the instances, and the heuristics we used. Section IV details the experimental setup (benchmark and algorithm settings), followed by the data analysis (Section V). Finally, Section VI wraps this work up, suggesting some paths for future developments.

II. FUNDAMENTALS

This section presents some key concepts about the Balanced Partition (BP) problem, hyper-heuristics, and MAP-Elites. We begin by describing what the BP problem is and the heuristics used to solve it. We then present hyper-heuristics and how they operate. We focus on selection hyper-heuristics since it is the approach we used in this work. After that, we explain MAP-Elites and the way we use it for optimizing a given objective function. We conclude this section by relating previous works done with MAP-Elites.

A. BALANCED PARTITION (BP)

Balanced Partition is a combinatorial optimization problem that requires balancing out the total load of two sets of elements. The problem starts with a set of items S where each item is associated with a weight. The task is to assign each item in S to one out of two subsets, S_1 and S_2, in such a way that their total weights become as close as possible. Thus, any item distribution represents a solution to the BP problem. The optimal solution is the one that minimizes the absolute difference between the sum of the weights of the subsets. There are two rules: the items can be assigned in any order, and each item can only be assigned to one subset.

To validate the quality of a solution, one may use a simple metric based on a quality indicator Q, such as

$$Q(S_1, S_2) \triangleq \left| \sum_{s_i \in S_1} s_i - \sum_{s_j \in S_2} s_j \right|. \quad (1)$$

Here, Q stands for the absolute value of the difference amongst set totals and, so, lower values are better. In fact, the ideal (minimum) value for Q is zero, but this may not be achievable in every instance.

Consider the following simple example. Imagine a problem instance given by $S = \{10, 20\}$. This problem has several solutions, where none, a single, or both items are moved into S_2. However, the 'best' solutions are either $S_1 = \{10\}, S_2 = \{20\}$ or $S_1 = \{20\}, S_2 = \{10\}$. In both cases, the sets are not fully balanced and $Q = |10 - 20| = 10$. Nonetheless, this is the lowest Q value that can be achieved and thus represents the best quality level. This metric also has the advantage of being extendable to multiple problem instances, where the average value of Q can be used for assessing the quality of a solver over a set of instances. If all instances are solved properly, the average Q value should be small.

There are two parameters that affect the hardness of a BP instance: the number of bits required for representing the item with the largest value and the number of items within the problem instance. According to Hayes [30], should their ratio be smaller than one, the instance is considered easy to solve, and perfect partitions can be expected to some degree. Should it be larger than one, the instance is identified as hard to solve, and it becomes unlikely to find perfect partitions. In the limit case, i.e., when the ratio is close to one, we can observe a phase transition phenomenon—an abrupt change in the probability of finding perfect partitions.

Now, there are several scenarios in real-life where resources must be evenly distributed. Therefore, the BP problem can be applied to a variety of contexts. Some common examples include scheduling and grouping. However, there are also moments when distributing the items into two sets is not enough. Hence, variations of the problem arise, such as the k-set Partitioning problem [31] and the Balanced Graph Partitioning problem [32], [33]. The former keeps the same overall structure but uses k subsets instead of two. The latter seeks to partition a graph into even-sized components while minimizing the number of edges. Figure 1 summarizes other problems that can be modeled as Balanced Partition, including: People Assignment [34], Routing [31], [35]–[38], Task Allocation [39], [40], File Placement [41], [42] and Scheduling [43]–[45]. We briefly describe them in the following lines.

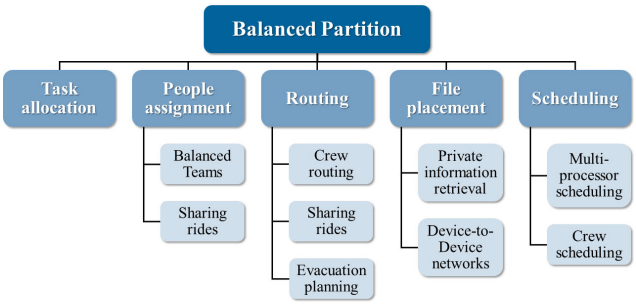


FIGURE 1. Some applications of the balanced partition problem that can be modeled as different problem domains.

The People Assignment problem has the objective of distributing a group of people with different capabilities or attributes in balanced subgroups or teams. This can prove useful when creating balanced teams [34] or when sharing rides [31]. The latter can also be modeled as a Routing Problem [31], [36], [37]. However, there are other applications, such as Delivery Routing or Evacuation Planning [38]. For instance, some authors have proposed a Balanced Graph Partitioning to reduce the overall computational complexity of the problem for Ride-sharing Routes [36], [37].

Subsequently, the Fair Task Allocation Problem consists of distributing n jobs to m workers [39]. Each job has a numerical weight depending on its complexity. The objective is to balance as much as possible the distribution of tasks among the workers. A more complex variation combines routing and task allocation: the Crew Scheduling and Routing Problem [35]. It consists of determining the best route and schedule for a single crew to accomplish the main job that can be divided into tasks at different locations.

In the case of the File Placement problem, the goal is to improve transmission and retrieval of information by distributing data in F equally sized packets. Zhu et al. tackled this problem for Private Information Retrieval context while minimizing file length [42]. Similarly, Wang et al. strove to characterize the placement and delivery of information in Device-to-Device networks [41].

Finally, and as an example of the application to Scheduling problems, one finds Multi-processor Scheduling. Here, the goal is to improve the parallel execution efficiency of tasks that share resources (e.g., external devices, shared memory, and files). To solve this problem, there exist multiple paradigms, such as a Partitioned Schedule [43], [45], a Global Schedule [46], and a Semi-partitioned Schedule [44].

B. HYPER-HEURISTICS (HHs)

Heuristics are common for solving challenging real-world Combinatorial Optimization Problems, mainly due to their computational simplicity. Such methods rely on problem-specific knowledge to operate and often produce time-cost efficient solutions. However, their performance can vary as they do not guarantee optimality.

Several approaches have emerged throughout the years to overcome the optimality issue. Hyper-heuristics are among the most recent ones. They aim to provide more generalized solutions by combining two or more heuristics as a problem is solved. In doing so, they match the advantages of solvers seeking to perform well over a set of problems instead of focusing on excellent results for a reduced subset of instances [15].

Hyper-heuristics can be classified with different criteria [15]. One of them leads to those that generate new heuristics and those that select among existing ones. The former does so by modifying or combining existing heuristics. The latter selects from a set of heuristics throughout an iterative search process. Nonetheless, the goal of both types of HHs is to improve the search process. Although both approaches are relevant, in this work, we focus on selection hyper-heuristics.

We are aware that many different types of selection HHs are described in the literature [16]. For example, there are HHs that work on batches of instances [47] and HHs that decide the next heuristic based on a reward/penalty strategy [48], [49] or on a threshold acceptance criterion [50], [51]. Among the different HHs described in the literature, two types are of particular interest for this work: rule-based HHs and sequence-based ones [3]. The former can be represented as a collection of rules, where a condition and an action make up each rule. The latter are more straightforward.

To establish appropriate conditions for a rule-based hyper-heuristic, we require a mapping for problem instances. This can be done, for example, by defining a set of features associated with the problem domain variables. For the BP problem, this can be given by the ratio of items belonging to one set. The action dictates the heuristic to be used when such a rule is triggered. Now, the decision about which rule is triggered usually falls to the rule closest to the current problem state [52]. Bear in mind that this model can be represented as a matrix, where rows stand for rules and columns, except the last one, represent feature values for the rule conditions. The final column corresponds to the action to use for the rules. Hence, this model is not limited to having a fixed number of rules per heuristic, which adds to its flexibility.

Sequence-based hyper-heuristics simply select heuristics by following a fixed sequence. Hence, they do not require any mapping of problem instances, although it may be incorporated to create hybrid approaches. Instead, their training seeks to find a proper heuristic ordering that maximizes performance over the set of instances.

Regard that the rationale behind this model is to identify a crucial sequence of heuristics. So, it is somewhat expected for sequences to be shorter than the number of choices to make when solving an instance. Because of this, it is also essential to consider how the sequence is cycled. Some not infrequent approaches found in the literature include the repetition of each element in the sequence, restarting the whole arrangement, and even mirroring the sequence [3]. Sánchez-Díaz et al. already analyzed the behavior of the first approach [53] and so, in this work, we focus on the other two. The authors calculated how many times the whole sequence required to be repeated for making all the decisions for a single instance. Then, they repeated each action in the sequence by said amount. The sequence is increased in backward order to account for non-integer repetitions. So, the last heuristics in the sequence may be repeated one more time than the first ones.

There are several works where authors have used sequence-based selection hyper-heuristics. Among the most relevant and recent ones, one may find the works of Kheiri and Keedwell [54], Kheiri *et al.* [55], and Kheiri [56]. They propose a sequence-based selection hyper-heuristic framework optimized with a hidden Markov Model to analyze and produce sequences of heuristics. They test this model with the Inventory Routing Problem [56] and the Nurse Rostering Problem [54], [55]. Ahmed et al. also apply this concept to solve the Urban Transit Route Design Problem [57]. A different approach was followed by Yates and Keedwell, where they use a logarithmic return to determine the sequences with the best performance [58]. These results demonstrate the potential of sequence-based selection hyper-heuristics for achieving the best performance and faster run times over other HHs.

C. MAP-ELITES (ME)

Multi-dimensional Archive of Phenotypic Elites (MAP-Elites, ME) is an *illumination* algorithm that was recently introduced [23]. An illumination algorithm is a tool for finding optimal solutions to a given problem. However, they differ from optimization algorithms since they seek to 'illuminate' the search space rather than exploit it. In this sense, an illumination algorithm focuses on exploration rather than exploitation. The reason for doing this is that it allows finding the optimal loci to a problem domain, but under

different design considerations, which can be particularly useful at the initial stages of a design process [59].

MAP-Elites has gained attention in the Evolutionary Computation community due to its simplicity and general applicability. The idea with ME is to map the highest-performing solutions (*elites*) within a discretized N-dimensional search domain (*feature space*), based on problem-specific features. These features are not necessarily correlated to the actual objective function. However, they describe some domain-specific properties of the candidate solutions, and the user is usually interested in seeing the effect of varying them. This search domain is also known as *behavior space* [23], [28], [60], but we opted for the first name in this work.

Through this mapping, the algorithm 'illuminates' the search space by showing the potential trade-off of each area w.r.t. the features of interest [29]. For example, when designing a robot like a walking gait, the space of possible behaviors can be described by how much each leg is involved in a gait, whereas the performance is measured by speed [28], [60]. So, one has an estimate of the maximal speed that the robot can reach for different leg involvement levels.

The benefits of MAP-Elites are diverse and rely mainly on the diversity of solutions given by the illumination of the solution space in terms of user-defined features. It gives the user insight into how combinations of characteristics correlate with performance [24]. According to Mouret and Clune [23], other benefits include:

- Improved optimization performance over some current state-of-the-art search algorithms;
- The search for a solution in a single cell is aided by the simultaneous search for solutions in other cells; and
- Returning a large set of diverse, high-performing individuals in a map can be used to create new types of algorithms or to improve the performance of existing ones.

For the sake of brevity, we now concisely mention the general idea behind MAP-Elites. A more detailed discussion of this algorithm is left for Section III, where we show how we adapted it for training sequence-based selection HHs. The algorithm initialization includes the generation of a set of random candidate solutions and their localization. To this end, each solution is evaluated and assigned to a cell within the grid (*feature space*). The best solutions within each cell are preserved. Afterward, an iterative process begins, encompassing the random selection of solutions for crossover and mutation operations. The offspring are placed within the grid. Should the cell be already occupied, both solutions are compared, and the best one (i.e., the *elite*) prevails. The final result is a grid that contains the best solutions for each cell.

Up to now, most applications of MAP-Elites have been restricted to robotic tasks. However, ME has also been used in applications such as: developing neural networks for computer vision tasks [23], tackling the Workforce Scheduling, and Routing Problem (WRSP) [24] and enhancing Genetic Programming [61]. To the best of our knowledge, there is only one case where MAP-Elites has been applied for constrained optimization [29]. In this regard, Fioravanzo et al. showed that ME cannot compete on all problems with state-of-the-art algorithms, such as those using gradient information or incorporating advanced constraint handling techniques. Nevertheless, they found that ME has a higher potential for finding constraint violations *versus* objectives trade-offs and providing new information about the problem. Hence, they state that ME could be an effective building-block for designing new constrained optimization algorithms.

III. PROPOSED APPROACH

In this work, we combine hyper-heuristics and MAP-Elites to solve the Balanced Partition problem. Our motivation is two-fold. First, we want to find optimal solutions to a set of problem instances. Second, we strive to provide a user with the effect of different parameter configurations and the trade-off between using two pools of heuristics. It is worth noting that we arrive at this idea after analyzing the work by Urquhart and Hart where they used ME for tackling the real-world workforce scheduling and routing problems [24]. However, we use ME for illuminating the space of heuristic combinations and thus focus on the hyper-heuristic model instead of the problem solution.

A. OVERVIEW

To link MAP-Elites within the hyper-heuristic model, we assign an integer for each heuristic, which acts as an ID. In this way, a number always represents the same heuristic (see Sect. III-C for details). Also, we represent a sequence-based hyper-heuristic as an array of such IDs. Each one of them indicates the heuristic to use at the corresponding step in the sequence. We use the term *cardinality* (C) for referring to the length (number of steps) of the hyper-heuristic [62]. For example, the array $\{1, 2, 1\}$ represents a sequence-based hyper-heuristic with $C = 3$, where the `Max` heuristic is used as the first and third steps and the `Min` as the second one.

With this link established, one can use ME to train sequence-based selection hyper-heuristics. Here, we assume that each dimension in the feature space belongs to one element in the sequence: a step. Moreover, the set of available heuristic IDs represents the domain of each dimension. Thus, the size of the search space that ME illuminates is H^{N_s}, where H is the set of heuristics and N_s is the number of steps in the sequence.

Since we expect sequences to be shorter than the total number of decisions required for solving the instance, there must be a way to reuse the sequence. We consider two approaches: restart and reflection. In the first case, the sequence begins anew. In the second one, the sequence repeats itself in a backward fashion. Bear in mind that, in practical terms, this corresponds to extending the sequence to cover the total number of decisions. Nonetheless, it does not affect the size of the search domain as the base sequence is already defined.

Figure 2a displays the effect of using one looping scheme or the other for a hyper-heuristic with $C = 3$. As one

may see, only the fourth decision changes. When using a restart scheme, the model uses heuristic h_4. But when using a reflection scheme, it employs h_3 instead. Even if this seems negligible, it will become more relevant as the sequence-based hyper-heuristic becomes longer and more diverse. Conversely, symmetric hyper-heuristics (e.g., $\{1, 2, 1\}$ and $\{2, 3, 4, 4, 3, 2\}$) lead to the same decisions regardless of the looping scheme. Also, note that the first C decisions are the same for both looping schemes, so excessively long sequences (e.g., those where C exceeds the number of decisions) will have the same performance.

Moreover, Figure 2b shows the relationship between a hyper-heuristic and the grid ME illuminates. For the sake of simplicity, we only show some dimensions. Particularly, we represent the third dimension by each layered map with transparency. In this way, what ME provides is an overview about the performance level that can be expected when selecting different values for the first two dimensions of the problem. In our case, this is done by selecting the best performance level across the remaining dimensions. Such a mapping has the added benefit of hinting at the effect that the first two dimensions hold over the performance of the model.

B. INSTANCES CONSIDERED IN THIS WORK

In this work, we use the sets of instances shown in Table 1. We generated such instances by randomly assigning integer values to a fixed number of items. Hence, the parameters that we consider for the generation are the number of elements within the instance and the number of bits required for representing its largest one. This allows us to create instances with different difficulty levels based on the ratio between the latter and the former.

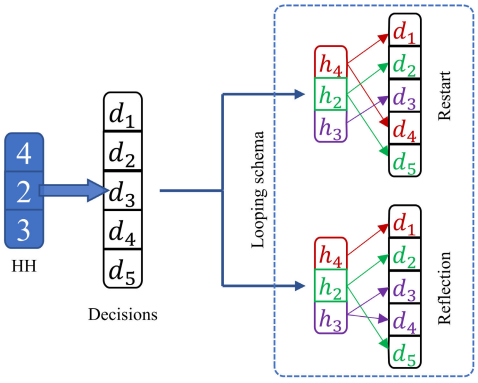
For this work, we label each set using the following pattern: Set-*numElements–numBits-numSet*. Note that whenever we omit the last number, it implies that there is only one set with the same number of elements and bits. So, Set-25-4 refers to the only set where instances have 25 elements and up to four bits per element.

TABLE 1. Sets of instances generated and used in this work. Instance Set name description found in Sec. III-B; NS: Number of sets with the given number of bits and elements; NI: Number of instances within each set. DI: Difficulty level of each instance within the set.

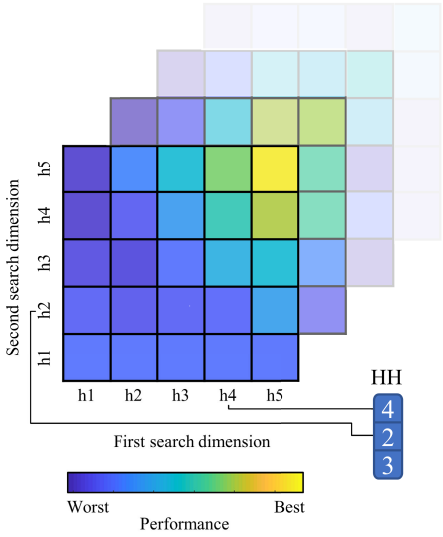
Instance Set	NS	NI	DI	Used in
Set-10-4	3	100	Easy	Exploratory experiments
Set-25-4	1	200	Easy	Initial and advanced experiments
Set-25-25	1	200	Neutral	Initial and advanced experiments
Set-25-50	1	200	Hard	Initial and advanced experiments
Set-40-4	1	200	Easy	Advanced experiments
Set-40-25	1	200	Easy	Advanced experiments
Set-40-50	1	200	Hard	Advanced experiments

C. HEURISTICS CONSIDERED IN THIS WORK

In this work, we implement a set of five simple heuristics representing problem-specific rules that provide computationally-efficient solutions for one or more problem



(a) Effect of looping scheme



(b) Link between Map-Elites and the model

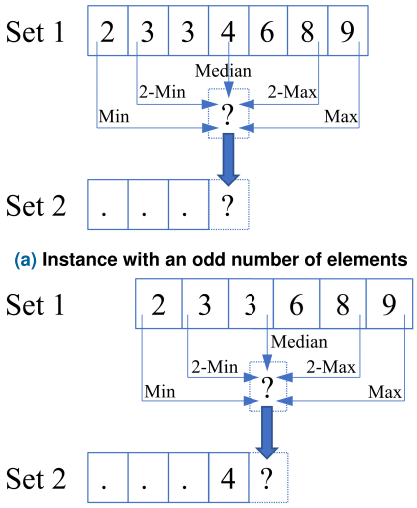
FIGURE 2. Some details about the proposed hyper-heuristic model.

instances. When solving an instance, we assume that all items are initially assigned to the first subset. Then, we progressively move them to the second subset (with each step of the solution) until it contains half or more of the total instance load. Remark that we do not allow items to return to the first subset to preserve a constructive approach in the solving process.

Even though there are several heuristics for solving partition problems, we select the following ones because of their simplicity: `Max`, `Min`, `2-Max`, `2-Min`, and `Median`. The former moves the largest item. `Min`, on the other hand, moves the smallest one. The following two heuristics, `2-Max` and `2-Min`, adhere to a similar procedure but instead select the second largest and smallest item, respectively. The remaining heuristic, `Median`, moves the middle element in the sorted instance. In the case of a sequence with an even number of elements, the previous element is selected.

Figure 3 displays a simple example of the items that each heuristic would select at two consecutive steps of the solution process. For the second step, the example assumes that

the previous item was selected with the `Median` heuristic. Each heuristic has a consecutive ID, starting at one and in the aforementioned order. So, ID three represents heuristic `2-Max`, for example.



(a) Instance with an odd number of elements

(b) Instance after choosing an element with the `Median` heuristic

FIGURE 3. Example of items selected for movement based on the different heuristics. For the sake of simplicity, we assume that the instance is already sorted.

D. ALGORITHM

We propose using a simplified MAP-Elites algorithm with no crossover, where each *genome* represents a sequence-based hyper-heuristic (Pseudocode 1). We adapt such a model from the one presented by Mouret and Clune [23]. To favor diversity, we force newly generated solutions to mutate into one different from their parent. Also, the average normalized quality level indicates the performance of a candidate hyper-heuristic (a *genome*), which we evaluate over a training set. This implies that one must divide the fitness Q achieved on each instance into the total load of the instance, as shown below:

$$Q'(Q,S) \triangleq \frac{Q}{\sum_{s_i \in S} s_i}, \tag{2}$$

where s_i represents each item within the set of items S, i.e., the instance. In using such a normalization, one can clearly identify the portion of an instance that remains unbalanced. In other words, it represents how close the sets are to a perfect split. A value of one indicates that all items are on one set (worst possible fitness). Conversely, a value of zero relates to a perfect split. This metric simplifies the comparison between instances with different parameters.

The algorithm starts by generating a multidimensional map with G elites, χ, and their performances P. Each of these elites is referred to as a *genome* and represents a candidate solution. As stated previously, in our case, a genome is a set of numbers (IDs) that represents a combination of heuristics, i.e., a sequence-based hyper-heuristic. For the remaining E-G iterations (where E is the total function evaluations), the algorithm selects a random genome and mutates it.

The mutation operator changes each element within the genome by a random heuristic ID based on a mutation rate. So, more than one element within the sequence can mutate in a given iteration. Also, bear in mind that the opposite may also happen, thus preserving the genome. Should this be the case, the algorithm selects a random element and changes it to a random alternative, which favors diversity. Then, it evaluates the resulting child.

Afterward, the child is assigned to the grid if the corresponding cell is empty. Otherwise, the algorithm discards the child since it represents a repeated solution. Bear in mind that, at this point, the solution should be compared against the one existing within the cell. Nonetheless, the discrete nature of our approach and the number of heuristics we considered make it possible for the map to be fully detailed. Hence, each cell represents a single solution. Therefore, if the cell is already occupied, there is no need to compare solutions as they are the same.

E. TIME COMPLEXITY ANALYSIS

We now present a brief complexity analysis for our proposal. To avoid overextending the manuscript, we have simplified it as much as possible. Let us begin by mentioning that the time complexity of our overall approach is given by $T_{\text{ME-HH}} = \mathcal{O}\{E * \#D * T_{\text{HH}}\}$, where E is the number of function evaluations, $\#D$ is the length of the instance dataset, and T_{HH} is the computing cost of a hyper-heuristic. This last term is proportional to the cardinality (length) of the hyper-heuristic, C, and the low-level heuristic with the highest computing cost, T_h. In a worst-case scenario, this same heuristic would be applied until solving the problem. So, $T_{\text{HH}} = \mathcal{O}\{C{*}n{*}T_h\}$, since n stands for the number of elements from the longest problem instance within the dataset D. The worst-case about solving the Balanced Partition Problem would be using the brute-force approach. It has a well-known time complexity for this problem of $\mathcal{O}\{2^n\}$ This result gives us the upper limit for our approach $T_{\text{HH}} < \mathcal{O}\{2^n\}$ since we did use simpler heuristics. Indeed, we must say that the implemented low-level heuristics has $T_h = \mathcal{O}\{1\}$ because they are just picking up values from an instance set. Summarizing, we get that $T_{\text{ME-HH}} = \mathcal{O}\{E * \#D * C * n\}$ corresponds to the time complexity of the proposed methodology.

IV. METHODOLOGY

We pursued the three-stage methodology summarized in Figure 4. First, we explored whether MAP-Elites provides a benefit when solving the Balanced Partition problem. To this end, we compared its performance against randomly generated sequences. Then, we analyzed the effect of different parameters upon the performance of MAP-Elites. Finally, we focused on verifying the performance of the algorithm under more demanding tests. We used randomly

Pseudocode 1 Simplified MAP-Elites Algorithm Used in This Work and Adapted From [23]

Require:
Training Parameters: Function evaluations E, number of initial genomes G, mutation rate μ_r, and instance dataset D
Hyper-heuristic Parameters: Steps within the sequence N_s, number of available heuristics N_h, and `CyclingScheme`
Ensure: Two maps, χ with the encoding of sequence-based hyper-heuristics and P with their performance

$\chi \leftarrow \emptyset$ and $P \leftarrow \emptyset$ ▷ Preallocate empty N_s-dimensional maps
for all $i \in \{1, \ldots, E\}$ **do** ▷ Repeat for all function evaluations
  if $i < G$ **then** ▷ Initialize using a random elite
    $HH' \leftarrow$ Random_Solution
  else ▷ Generate new solutions from elites within the map
    $HH \leftarrow$ Random_Selection(χ) ▷ Randomly select an elite HH from the map χ
    $HH' \leftarrow$ Mutate(HH) ▷ Create HH' by mutating HH
  $C \leftarrow \#HH$ ▷ Read the current hyper-heuristic cardinality or number of steps
  $p' \leftarrow$ Evaluate_HH(HH') ▷ Assess the performance of the hyper-heuristic
  if $\chi(HH') == \emptyset$ **then**
    $P(HH') \leftarrow p'$ and $\chi(HH') \leftarrow HH'$ ▷ Store the solution and its performance
return χ, P

procedure Mutate(x) ▷ Mutation procedure for a given genome
  $x' \leftarrow x$ ▷ Copy the original genome
  for all $x'_i \in x'$ **do**
    if $\mathcal{U}(0, 1) < \mu_r$ **then** $x'_i \sim \mathcal{U}\{1, N_h\}$ ▷ Change component i-th by a random heuristic ID
  if $x == x'$ **then**
    $i \sim \mathcal{U}\{1, C\}$ ▷ Select a random component of x'
    $x'_i \leftarrow$ Random_Change(x'_i) ▷ Change x'_i to a different heuristic ID
  return x'

procedure Evaluate_HH(x, D) ▷ Hyper-heuristic evaluation over a given dataset
  $p_N \leftarrow 0$
  for all $d_i \in D$ **do**
    $p \leftarrow$ Run_HH(x, d_i) ▷ Assess the HH performance over the instance d_i
    $p_N \leftarrow p_N + Q'(p, d_i)$ ▷ Accumulate the normalized performance using Equation 2
  return $p_N / \#D$ ▷ Return the average performance by dividing into the dataset length

procedure Run_HH(x, d) ▷ Implement the hyper-heuristic x to solve the instance d
  $S_1 \leftarrow d$, $i \leftarrow 1$, and $i_{\text{prev}} \leftarrow 1$ ▷ Assign all items to the first set and initialize the step counters
  while Status(d) == 'unsolved' **do**
    $S_1, S_2 \leftarrow x_i\{S_1\}$ ▷ Move an item to the second set using the i-th heuristic
    $i, i_{\text{prev}} \leftarrow$ Next_Step(i, i_{prev}, `CyclingScheme`) ▷ Increase the step counter according to the cycling scheme
  return $Q(S_1, S_2)$ ▷ Return the performance evaluated using Equation 1

procedure Next_Step(t, t_{prev}, `CyclingScheme`) ▷ Determine the next step value
  $t_{\text{curr}} \leftarrow t$
  if `CyclingScheme` == 'reflection' **then** ▷ Reflection scheme, e.g., it is 1, 2, 3, 3, 2, 1, 1, 2, ..., for $x = \{1, 2, 3\}$
    if $(t == 1)\&(t == t_{\text{prev}})$ **then** $t \leftarrow t + 1$
    else if $1 < t < C$ **then** $t \leftarrow 2\,t - t_{\text{prev}}$
    else if $(t == C)\&(t == t_{\text{prev}})$ **then** $t \leftarrow t - 1$
  else if `CyclingScheme` == 'restart' **then** ▷ Restart scheme, e.g., it is 1, 2, 3, 1, 2, 3, 1, 2, ..., for $x = \{1, 2, 3\}$
    if $t == \#x$ **then** $t \leftarrow 1$, **else** $t \leftarrow t + 1$
  return t, t_{curr}

generated instances throughout all the stages, as mentioned in Section III-B.

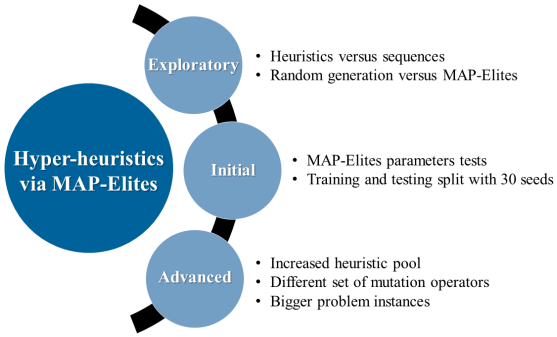


FIGURE 4. Three-stage methodology followed throughout this work.

A. EXPLORATORY TESTS

In these tests, our goal was two-fold. We strove to estimate whether a combination of heuristics yields better performance than any standalone heuristic. Also, we sought to detect if MAP-Elites is a better alternative over randomly selected sequences. With this, we aimed at laying out the feasibility of using a sequence-based selection hyper-heuristic powered by MAP-Elites. So, we split this stage into two phases that use Set-10-4-1 to Set-10-4-3.

1) HEURISTICS VERSUS SEQUENCES

To have reference values, we solved the instance sets with two base heuristics: `Max` and `Min`. Then, we analyzed the feasibility of combining heuristics by using the simplest sequence-based hyper-heuristics: {`Max, Min`} and {`Min, Max`}. The remaining combinations (i.e., {`Min, Min`} and {`Max, Max`}) can be disregarded as they consist of a repeated heuristic, which is the same as the standalone version (i.e., `Min` and `Max`, respectively). As we mentioned in Section III-D, the performance of each set was assessed by the average of the normalized metric shown in Equation 2.

Afterward, we studied whether longer sequences are better than shorter ones. To this end, we generated random sequences of the base heuristics (i.e., `Min` and `Max`) with 3, 5, 10, and 15 steps, and applied them to the same sets of instances. Since the next step involves a comparison against the MAP-Elites training, we selected the best random sequence among a set of 10 candidates striving for a fair comparison. We based this selection on the mean fitness criteria of all ten sequences, and repeated the process 50 times to account for randomness.

2) RANDOM GENERATION VERSUS MAP-ELITES

The main difference between MAP-Elites and random sequences falls to the mutation operator. Therefore, this test tackled the justification of such an operator. This time, hyper-heuristics were trained with MAP-Elites, considering a mutation rate of 0.3 and 3 initial genomes. To keep the comparison fair, MAP-Elites ran for seven iterations, so the total function evaluations remain at 10. The other parameters were preserved. We also generated hyper-heuristics with 3, 5, 10, and 15 steps and repeated each experiment 50 times.

B. INITIAL TESTS

Since it seems feasible for MAP-Elites to train a sequence-based hyper-heuristic (see Section V-A), in this stage, we delved deeper into its analysis. Parting from datasets Set-25-4, Set-25-25, and Set-25-50, we generated subsets of training and testing instances with 30 different seeds to avoid bias due to the random effects. In this way, we trained sets of varying difficulty, where each instance has 25 elements. The seeds used for each split can be found at http://github.com/iamaya2/CombinatorialProblemInstances. For this testing stage, we trained hyper-heuristics, considering the following parameters:

- Number of steps in the sequence (N_s): 10
- Number of initial genomes (G): 6
- Mutation rate (μ_r): 0.3
- Available heuristics ($\mathfrak{H}$): {`Max, Min`}
- Number of function evaluations (E): 50
- Cycling Scheme (`CyclingScheme`): 'restart'
- Number of repetitions (runs) per experiment: 50
- Training ratio: 50%

Then, we used each sequence to solve the testing subsets and compare their performance. Afterward, we investigated how the following parameters affect hyper-heuristic performance: `CyclingScheme`, N_s, G, and μ_r. For each test, we kept the other parameters on the previously stated values.

1) EFFECT OF CYCLING: RESTART VERSUS REFLECTION

Up until now, all the tests were performed using sequence-based hyper-heuristics that restart the sequence. Hence, at this point, we compared the effects of using a different cycling scheme. So, we repeated the previous experiments (i.e., we employed the same seeds for splitting the sets), but we now trained hyper-heuristics with a reflection-based (`CyclingScheme`) scheme.

2) EFFECT OF CARDINALITY

To better understand the effect of cardinality, we changed the number of steps in the sequences to some arbitrarily selected values between 5 and 20. Our objective was to study if there is a pattern in the performance of shorter and longer sequences. Moreover, we carefully assessed the hyper-heuristics performance when their cardinality equals half the instance length (cardinality). The reason for doing so is that our instances are randomly generated through a uniform distribution. Hence, we expected that half the items must be moved before achieving balance. So, we tested cardinality values of 5, 10, 12, 13, 14, 15, and 20.

3) EFFECT OF INITIAL GENOMES

To test the effect of initial genomes, we set the number of function evaluations to 50. Then, we implemented nine different ratios between 10% and 90% for setting the number of initial genomes between 5 and 45. Here, we expected that hyper-heuristics with large values perform similarly to a random selection, as ME would only have a few iterations for improving.

4) EFFECT OF THE MUTATION RATE

The mutation in the algorithm provides diversity in the solutions from one generation of the population to the next one. As explained in Section III-D, this increases the chance of occupying new cells in the grid by trying more possible solutions. It is important to highlight that a very low mutation rate could hinder the ability of the algorithm for diversifying the solutions. While, if the mutation rate becomes excessively high, it may transform our approach into the simple random search used in the exploratory experiments. Hence, the need for determining the precise effect of such a parameter. In this experiment, we varied the mutation rate between 0.0 and 1.0 with increments of 0.1.

5) EFFECT OF THE TRAINING RATIO

For the prior tests, we split each dataset from Table 1 into different training and testing subsets. However, having an excessively high training ratio is undesirable as it makes the process more computationally intensive. Unlike, training with too few samples could hinder performance as not all kinds of instances are represented. Both of these cases should be avoided as they lead to unsought behaviors [63]–[65]. In this phase of experimentation, we aimed at testing the effect of training with a different number of instances. In this way, we explored the sensitivity of the proposed model and determined a feasible level for future usage. So, we considered the following training ratios: 10%, 30%, 50%, 70%, and 90%.

6) ORACLE

As a final point of comparison at this stage, we looked forward to assessing how the hyper-heuristics fare when matched against a more robust solver. To do so, we procured a synthetic solver commonly known as the oracle. Such an oracle is able to perfectly select the best standalone heuristic for each instance. Since this approach is unfeasible to train, we deployed a brute-force process where we run each heuristic and then extracted the best result. For these tests, we trained five fresh hyper-heuristics per seed, using the best parameter values obtained from the previous phases (see Section V for the details), such as:

- Number of steps in the sequence (N_s): 15
- Number of initial genomes (μ_r): 15
- Mutation rate (G): 0.4
- Available heuristics ($\mathfrak{H}$): {`Max, Min`}
- Number of function evaluations (E): 50
- Cycling scheme (`CyclingScheme`): 'restart'
- Training ratio: 50%

C. ADVANCED TESTS

The last stage of experiments aimed to evaluate the algorithm under more exacting tests, distributed throughout three phases. So, we analyzed how performance changes by modifying the quantity of available heuristics and changing the nature of mutation operators. We also checked if the algorithm is sensitive to the size of the instance by training hyper-heuristics with larger sets.

1) EFFECT OF INCREASING THE POOL OF HEURISTICS

First, we added three heuristics to the search domain of the algorithm: `2-Max, 2-Min` and `Median` (see Section III-C). In doing so, we amplified the search domain. For example, if we consider sequences with the same cardinality (i.e., $C = 15$), the domain increases by almost one million heuristic combinations. This, in turn, extends the memory requirements of the proposed approach beyond our capabilities. So, to keep things manageable, we reduced the cardinality to $C = 10$, which only increased memory requirements by a factor of 300. So, the tests from this stage utilized the same datasets from Section IV-B but with the following parameters:

- Number of steps in the sequence (N_s): 10
- Number of initial genomes (G): 15
- Mutation rate (μ_r): 0.4
- Available heuristics ($\mathfrak{H}$): {`Max, Min, 2-Max, 2-Min, Median`}
- Number of function evaluations (E): 50
- Cycling scheme (`CyclingScheme`): 'restart'
- Number of repetitions (runs) per experiment: 50
- Training ratio: 50%

Moreover, we assessed how the oracle shifts by repeating tests from Section IV-B6 with the extended heuristic set.

2) EFFECT OF MUTATION OPERATORS

For the second phase, we changed the mutation operators. So, we no longer employed those described in Section III-D but a set of five operators that alter the hyper-heuristic. We strove to use more disruptive mutation operators, to test their effect in algorithm performance. So, at each function evaluation, one of these operators was randomly selected and applied as described below. To better illustrate the inner workings of each operator, consider the following set of available heuristics $\mathfrak{H} = \{h_1, h_2, h_3, h_4, h_5\}$ and the hyper-heuristic given by $HH = \{h_4, h_5, h_2, h_1, h_3\}$, which mutation targets.

Single-Point Flip selects a position $i \in \{1, \ldots, C\}$ and changes the element for any of the available heuristics (selected at random). For example, let us assume that $i = 3$. Since the new heuristic can be any $h_h \in H$, let us consider that h_1 is selected. Then, the mutated hyper-heuristic becomes: $HH' = \{h_4, h_5, h_1, h_1, h_3\}$.

Neighbor-based Single-Point Flip selects a position $i \in \{1, \ldots, C\}$ and changes the element by a copy of a randomly selected neighbor. Again, let us assume that $i = 3$ as an example. Then, the new heuristic can

be either h_5 or h_1. Let us suppose that the former is selected. Then, the resulting hyper-heuristic is $HH' = \{h_4, h_5, h_5, h_1, h_3\}$.

Neighbor-based Two-Point Flip is a variant of the Neighbor-based Single-Point Flip in which two positions $i, j \in \{1, \ldots, C\}$ copy one of their neighbors. Let us suppose that $i = 1$ and $j = 3$. In the first case, the available heuristics are $\{h_3, h_5\}$; do note that the operator wraps around the sequence. In the second one, they are $\{h_1, h_5\}$. Imagine that h_5 and h_1 are selected, respectively. Then, the hyper-heuristic becomes $HH' = \{h_5, h_5, h_1, h_1, h_3\}$.

Single-Point Swap interchanges heuristics at two randomly selected locations, $i, j \in \{1, \ldots, C\}$. Let us assume that $i = 1$ and $j = 3$. Thus, the resulting hyper-heuristic is $HH' = \{h_2, h_5, h_4, h_1, h_3\}$.

Two-Point Swap variant of the Single-Point Swap where two pairs of elements are interchanged, i, j and k, l, $\forall i, j, k, l \in \{1, \ldots, C\}$. Let us assume that $i = 3, j = 5$, $k = 2$, and $l = 4$. Thus, the resulting hyper-heuristic is given by: $HH' = \{h_4, h_1, h_3, h_5, h_2\}$.

Bear in mind that any of these operators can replace the corresponding heuristic with the same one. Also, one may include more operators, but we limited ourselves to these five for the sake of simplicity. Moreover, we used the same sets from Section IV-B and the parameters from the previous phase (Section IV-C1) to compare data with the previous operators.

3) EFFECT OF SETS WITH MORE ELEMENTS

Finally, and to ascertain the performance of the algorithm for larger instances, we selected Set-40-4, Set-40-25, and Set-40-50 for training new hyper-heuristics. Remember that, as shown in Table 1, these sets have 40 elements per instance, and preserve the number of bits (4, 25, and 50). So, we still have instances with varying difficulty levels.

V. RESULTS

We present the experimental data generated at each testing stage. To facilitate things for the reader, we preserve the structure followed in Section IV.

A. EXPLORATORY TESTS

Since this stage contains two phases with comparable data, we condense the information into Table 2. This way, one may carry out comparisons across the different models more easily. Notice that we provide the normalized mean fitness for each set, as well as an overall mean fitness.

1) HEURISTICS VERSUS SEQUENCES

Our data show that solving a set of instances with low-level heuristics can result in a similar mean fitness value. However, the opposite may also happen. Consider Table 2: For the first set (Number of Set, NS, $= 1$), the difference in heuristic performance is marginal (0.006 units). However, for NS $= 3$, the gap rises to 0.037 units, representing a sixfold increase. This evidences the variability of heuristics when tackling problem instances. It also supports the need for developing instance generators that can be tailored to specific heuristics, such as the ones shown in [66]–[68].

Nonetheless, a stimulating effect appears when combining both solvers, independently of the performance gap previously exhibited. If heuristics are combined following the {`Max, Min`} sequence, the hyper-heuristic solves all sets more efficiently and reduces the mean fitness in about 15% of the best heuristic. But, using the opposite combination, i.e., {`Min, Max`}, actually worsens all solutions, increasing the mean fitness in about 23% beyond the worst solution. This implies that combining heuristics can be better than using a single one. However, it requires additional processing for finding a proper combination.

There are several ways to obtain an improved heuristic combination. Nevertheless, for now, we want to determine if randomly selecting heuristics is good enough. As can be seen in Table 2, using longer sequences proves advantageous for performance. Even combinations of three heuristics allow finding the right configurations that outperform sequences with $C = 2$ in about 20%.

Even so, such sequences seem to fall short as longer sequences exhibit better performance, even on a per-instance base, as Figure 5 shows. Nonetheless, cardinalities beyond $C = 10$ seem to lead to no further improvements. This is somewhat expected since instances contain ten elements, and so no more than ten decisions are to be made. Hence, going beyond this value does not make much sense. Still, we added this scenario to validate the sequence-based model functioning and determine whether the additional number of combinations hinders the sequence performance.

There is, in fact, an interesting phenomenon. Throughout each test, we performed ten function evaluations. Plus, there are 8 different sequence-based hyper-heuristics with $C = 3$ and 32 with $C = 5$. So, it is likely that the algorithm finds the best combination for $C = 3$, whilst this may not be the case for $C \geq 5$. Nonetheless, the former was outperformed by the latter. Consequently, there seems to be a benefit at the individual (instance) scale when using longer sequences. As Figure 5a shows, using sequence-based hyper-heuristics with $C \geq 10$ allows finding solutions with about 10% less discrepancy. Nonetheless, variability increases.

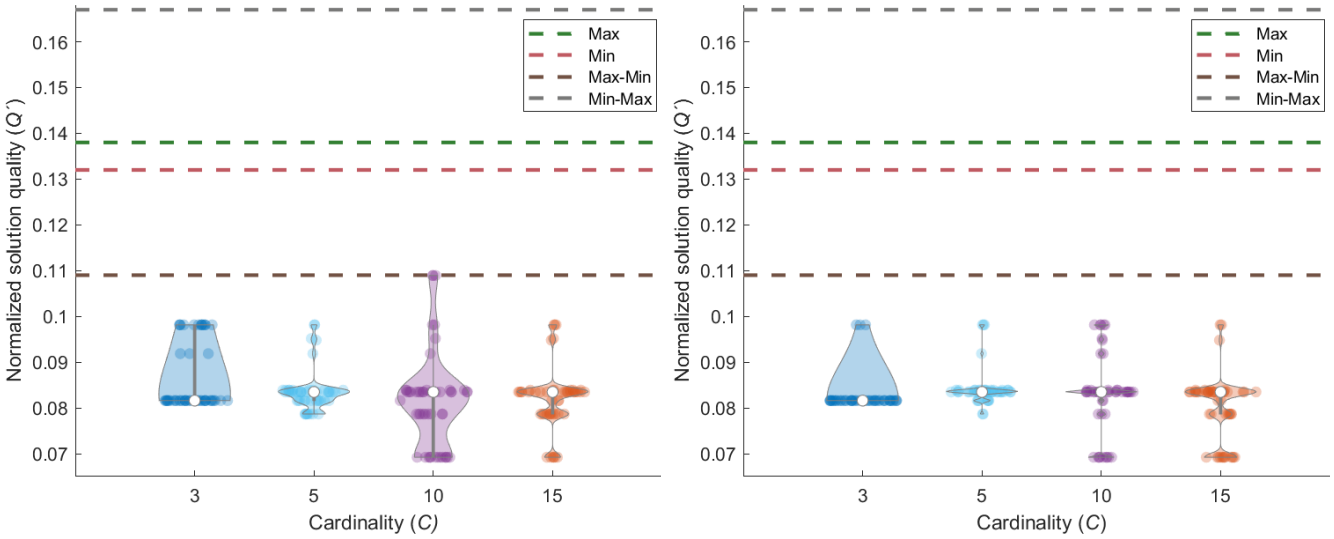
We may expect the previous result since the number of combinations increases significantly (1024 for $C = 10$). So, finding a good combination by selecting among ten randomly generated sequences does not provide steady results. One may improve upon this by implementing a more elaborate method for finding proper sequences, as is shown in the next phase.

2) RANDOM GENERATION VERSUS MAP-ELITES

As with random sequences, ME outperforms standalone heuristics and sequences with $C = 2$ (Table 2). Moreover, the behavior of hyper-heuristics for corresponding lengths of

TABLE 2. Performance achieved by different solvers over the three sets of generated instances with 10 elements and 4 bits per element (Set-10-4-1 to Set-10-4-3). Each result represents the normalized mean fitness over the whole dataset and averaged across 50 runs. NS: Number of the set being used. SBHH: Sequence-based hyper-heuristic. C: Cardinality as described in Section III-A. Best values (per row) are highlighted in bold.

NS	Performance achieved by: Single heuristic		SBHH with $C = 2$		SBHH trained with random sequences				SBHH trained with MAP-Elites			
	Max	Min	Max-Min	Max-Min	$C = 3$	$C = 5$	$C = 10$	$C = 15$	$C = 3$	$C = 5$	$C = 10$	$C = 15$
1	0.138	0.132	0.109	0.167	0.087	0.084	0.081	0.082	0.083	0.084	0.083	**0.080**
2	0.113	0.129	0.094	0.178	0.074	0.071	0.068	0.067	0.074	0.067	0.069	**0.064**
3	0.118	0.155	0.107	0.167	0.083	0.079	**0.075**	0.076	0.080	0.077	0.076	0.078
Mean	0.123	0.139	0.104	0.171	0.081	0.078	0.075	0.075	0.079	0.076	0.076	**0.074**



(a) Using random sequences

(b) Using MAP-Elites

FIGURE 5. Performance of hyper-heuristics with different sequence lengths, for Set-10-4-1. Each dot represents the best fitness after 10 function evaluations for each of the 50 runs. The dashed lines represent the performance achieved by the available heuristics, and by all sequences with a cardinality of two.

both approaches is similar. However, this time performance stabilizes for $C \geq 5$. Nonetheless, it seems as if ME allows for better performance, as indicated by the lower mean value across all sets. In fact, ME got better results in 58.33% of the 12 scenarios (3 datasets, 4 cardinality values). Besides, for two out of the three sets, the best performing hyper-heuristic corresponds to one trained with ME, which relates to the one with the best mean fitness across all sets.

To deepen upon the behavior of ME on a per-instance basis, let us analyze the corresponding violin plots (Figure 5b). The one with three steps exhibits the worst performance, although most of the runs achieved the same fitness value. Note that the random sequences also achieve this performance, as anticipated, but they do not exhibit the same consistency as ME. It is expected that all sequences with $C = 3$ are generated at least once throughout all random sequences. Hence, we may conclude that such a value represents the best result that can be achieved (under these conditions) for such a cardinality value. Besides, ME is better at finding it.

Sequences with $C = 10$ and 15 exhibit a similar performance due to the lengths of the instance and of the sequence. Remember that we are looking to split an instance with ten elements into two balanced subsets. There does not exist a case where all pieces are moved to the second subset. Consequently, a sequence with a cardinality value beyond the number of elements within the instance would not execute any extra steps.

These findings support the notion that there is a relationship between the cardinality of sequence-based hyper-heuristics and the number of elements within the instance. The ideal ratio between them is not necessarily one, as this depends on the complexity of the problem, the difficulty of the instances being solved, and the diversity among the heuristic performance. For the instances analyzed in this section, i.e., 10 elements and 4 bits each, a ratio of 0.3 seems too low. Due to the exploratory nature of this section, we do not run more detailed tests, but it seems that a ratio between 0.5 and 1.0 works best. Also, any sequence-based

hyper-heuristic longer than the instance is unnecessary since the extra steps would not be executed and increases the complexity of the model and, thus, the time required for training it.

Finally, it is essential to highlight that ME seems to improve the generation process stability. The more compact violins evidence this in Figure 5b, which implies a smaller standard deviation. Hence, using ME for training the sequence-based hyper-heuristics is justified as it provides a way of obtaining equal or better results in a more repeatable way. So, we now move on to analyzing the proposed approach in more detail.

B. INITIAL TESTS

Now that we have asserted the feasibility of using sequence-based hyper-heuristics powered by MAP-Elites, we assess their behavior under different scenarios. So, we use 30 different seeds for splitting each dataset into proper training and testing subsets, with varying degrees of difficulty.

Figure 6 shows the ME performance on all splits of every set. Each data point is given by the best performance achieved by ME after training for 50 function evaluations. Moreover, violins contain 50 points, which represents each of the 50 runs for the corresponding split. As explained in Section IV-B, we normalize data to facilitate comparisons and provide a metric less biased by the size of items within an instance. As one may see, all splits include some runs where the trained sequence-based hyper-heuristic performs poorly. However, they represent the outlying data.

Conversely, most data exhibit a median value that oscillates within a defined range. In easy instances (namely, Set-25-4), this range is given by $[0.024, 0.028]$. Although there are some splits (such as the first one) where several runs flaunt an improved performance, there is also a behavior worth noticing. When migrating to more difficult instances, such as those in Set-25-25 and Set-25-50, the range for the median value shifts upwards, as expected. Nonetheless, this range becomes the widest at the limit case, where the number of bits per element equals the number of items within the instance. This may indicate that instance behavior may relate to such parameters, and thus one could use them for predicting the behavior of unseen instances. Notwithstanding, this exciting phenomenon escapes our scope, and so we do not deepen upon it.

Let us now analyze the behavior of these hyper-heuristics over unseen instances. To this end, Figure 7 summarizes the median performance over each training and testing subsets. There are some elements worth noticing here. First, the aforementioned behavior where the limit case has the widest performance range is also evident here. Also, it is a tendency for achieving a better performance in the training set. Although, this does not happen for every split (subset) of each dataset. For example, with Set-25-4, the fitness on the testing set was better than on the training set for 36.7% of the splits. Similarly, this ratio was 23.3% for Set-25-50. Even so, once again Set-25-25 (the limit case) exhibits a different

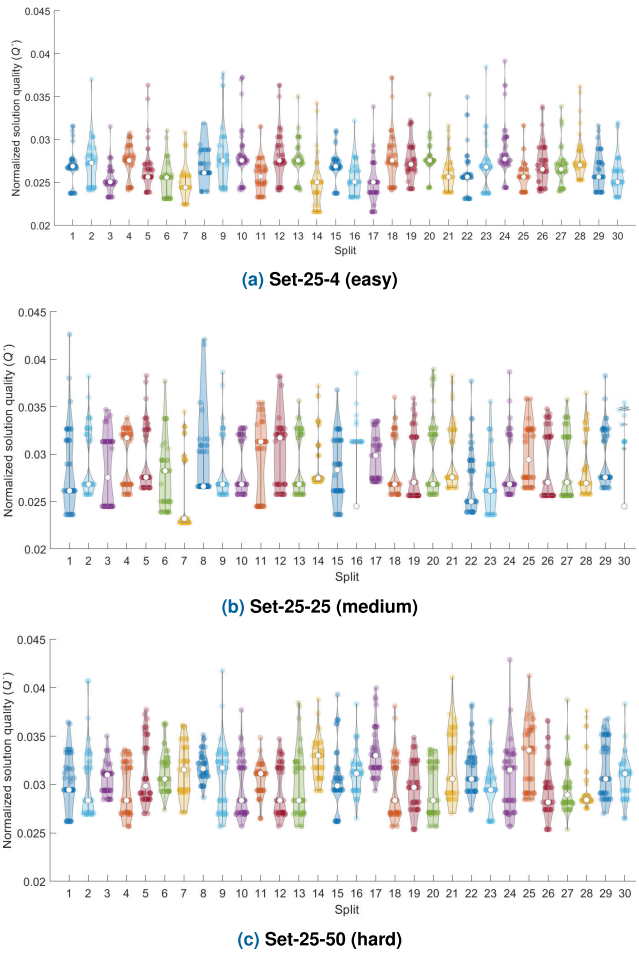
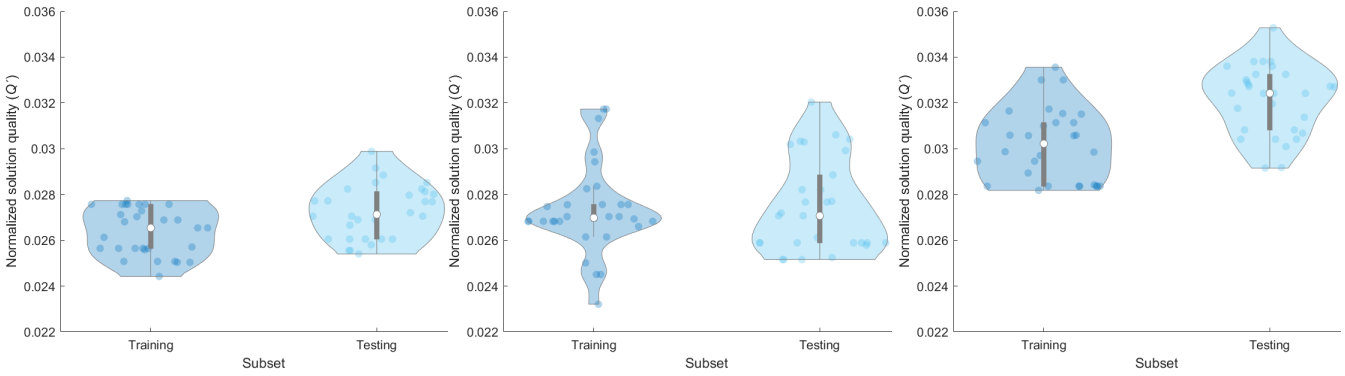


FIGURE 6. Performance achieved by sequence-based hyper-heuristics ($C = 10$) trained with MAP-Elites (ME) over different training splits of the datasets. Each dot represents the best fitness achieved after 50 function evaluations. Each violin contains 50 points, representing all runs for that split.

tendency, one where the median performance is virtually the same for training and testing. But, one where performance on the test was better for over half the splits (56.7%).

It is also crucial to examine the way in which the solution evolves for different conditions. So, we plot the distribution of fitness across iterations. This leads to a total of 90 figures, because of the 30 splits for each dataset. For obvious reasons, it is unfeasible to show them all here. Instead, we only include some illustrative examples. Figure 8 shows an example of a good, an average, and a bad split for instances within Set-25-4. As we can observe, instance splitting affects performance. In some cases (e.g., Figure 8a), hyper-heuristic generation becomes relatively stable, up to the point where even the worst repetition exhibits a performance quite close to the median one. Alas, in other cases the opposite happens and some repetitions stagnate early on (Figure 8c). Nonetheless, and even when this happens, only a couple repetitions exhibit such a behavior. Thus, a minimum of 90% of all repetitions (for each split) exhibit a good performance.



(a) Set-25-4 (b) Set-25-25 (c) Set-25-50

FIGURE 7. Performance achieved by sequence-based hyper-heuristics ($C = 10$) trained with MAP-Elites over training and testing subsets of Set-25-4, Set-25-25, and Set-25-50. Each violin contains 30 dots, where each one represents the median performance over 50 runs of a different subset.

If we now focus on the best performance, there are also some extraordinary elements. For example, there is at least one repetition for all splits that ends up having a solution with a fitness value below the 0.025 mark. There also are four splits for which the best repetition yielded the best solution right from the start. In one of those splits, such a value sits below the 0.020 mark, representing one of the best values achieved by all splits. Of course, this is only natural as one would expect that at one moment or the other, random number generation provides a good enough solution, as is familiar with stochastic methods such as metaheuristics. Finally, regarding the evolutionary process of the solutions, the number of iterations seems reasonable, since the median behavior for all cases behaves asymptotically towards the end.

There are some common elements regarding the evolution of ME in the remaining datasets. For example, they both have good (Figures 8d and 8g), regular (Figures 8e and 8h), and bad runs (Figures 8f and 8i). Furthermore, as problems become more difficult (i.e., the number of bits rises), the best solution impoverishes, as expected. Even so, such value does not rise too much. For instances with 25 bits, all splits included a run where the fitness sat below the 0.027 mark. This value grows to 0.031 for the most challenging instances (50 bits). Despite of this, the best solution across all runs and splits is quite similar for both sets of instances, as it hovers around the 0.025 mark. Again, there are some splits where the the algorithm achieved the best value due to the initial randomness (6 and 3 for 25 and 50 bits, respectively). Moreover, there are splits where the median performance behaves asymptotically towards the final function evaluations, though some may benefit from some additional evaluations. This is understandable, as these problems are more complex than those with 4 bits.

1) EFFECT OF CYCLING: RESTART VERSUS REFLECTION

So far we have run tests considering a restart cycling scheme. Thus, it is relevant to analyze the effect of reflecting the sequence. As Figure 9 shows, the algorithm virtually retains training performance. However, the overall performance in the testing sets significantly decreases when using the reflection cycling scheme. Previously, the median performance for training and testing subsets was similar. Now, the median fitness values for the testing subsets are about twice those for training. Moreover, there is no single split where hyper-heuristics perform better on the testing subset than on the training one, as it is evident from Figure 9. In contrast, the previous cycling scheme (i.e., restart) allows for such an improvement for 23% of more of the splits. All of this implies that the reflection looping scheme has lower generalization capabilities. So, we certainly do not encourage its use, at least for this particular problem. However, we do not discard the possibility that it may become a good alternative in other domains; this must be thoroughly explored in future works.

There also are some exciting elements regarding the fitness evolution of this cycling scheme. For some splits, e.g., the fourth one from Set-25-4, the behavior (Figure 10a) is virtually the same as before (cf. Figure 8a). This corroborates similar trends in the training scenarios. Others, however, alter their behavior a bit more, although not that much. For example, the split shown in Figure 10b exhibits a poor performance for the restart cycling scheme (Figure 8f). But, for the reflection one, it stands at a medium level, since even the worst run improves to a good value; though it does so towards the end of the iterations. Moreover, the worst run is about 0.01 units better than for the restart scheme. Conversely, the split shown in Figure 10c used to have a medium performance (cf. Figure 8e), but changing the cycling scheme lowers it to a poor one. Nonetheless, variations scarcely go beyond these margins, and there are only slight changes in the behavior, thus preserving the overall shape of fitness evolution.

2) EFFECT OF CARDINALITY

Our data reveal that the cardinality (sequence length) of a hyper-heuristic indeed affects its performance (Figure 11). We briefly showed this in the exploratory tests, where we

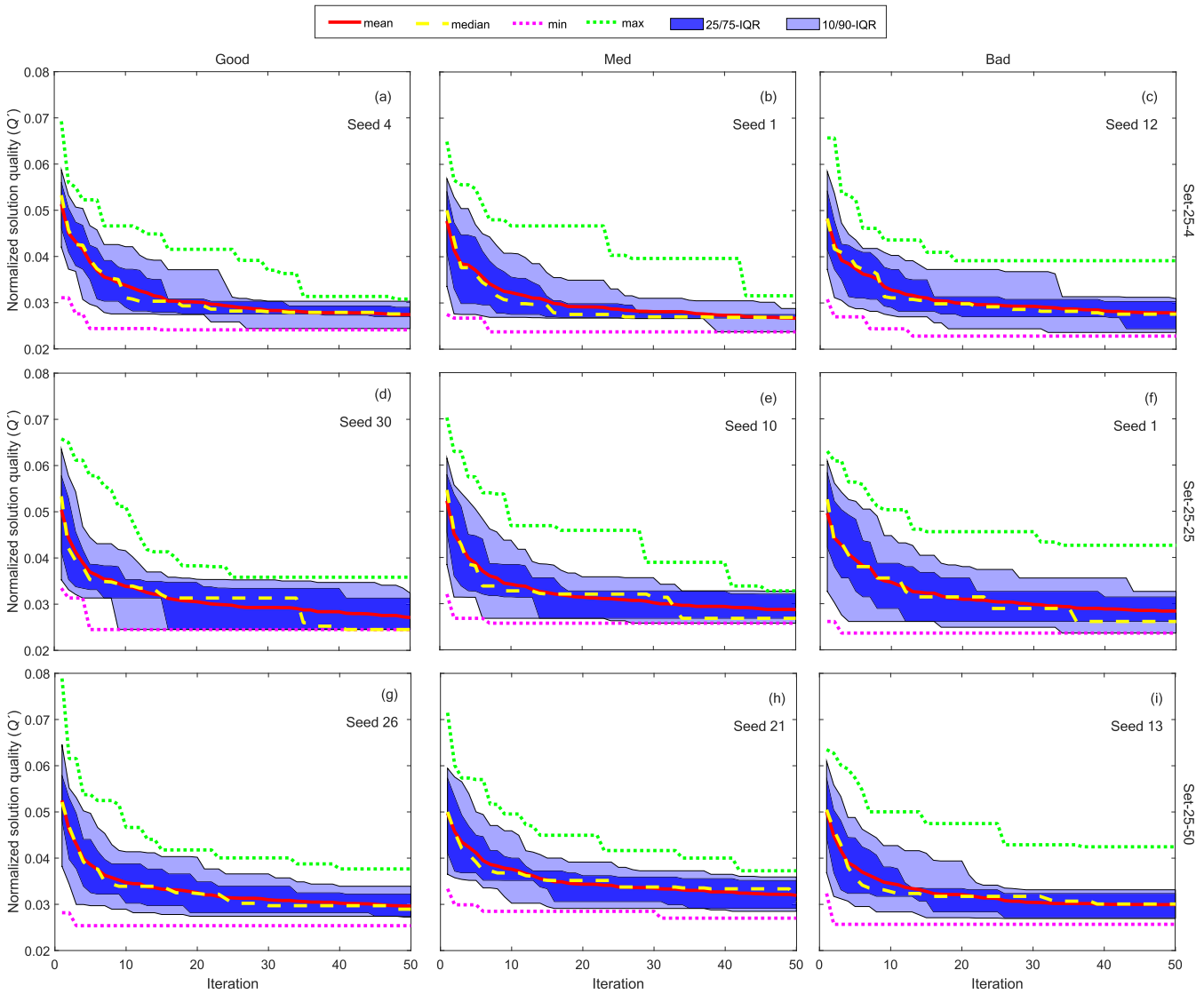
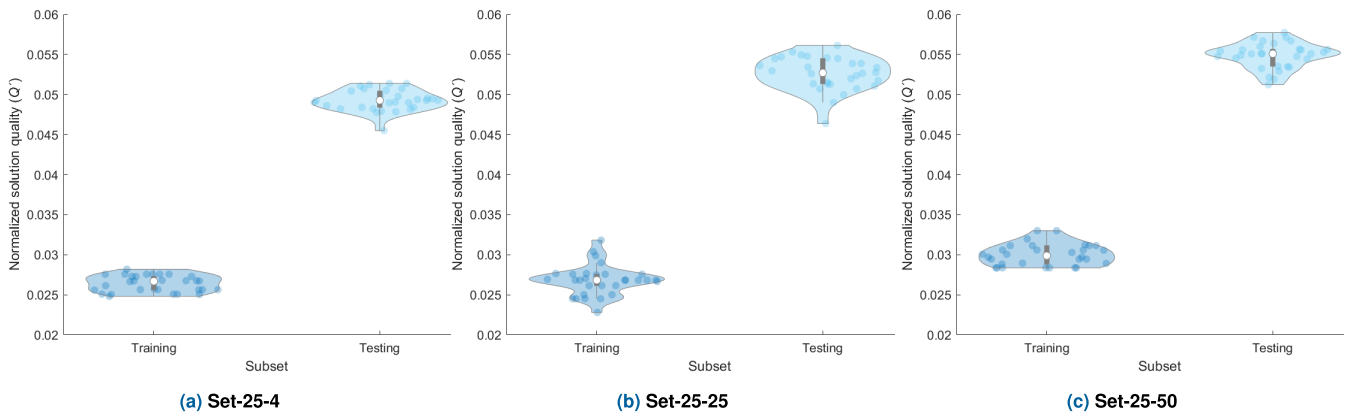


FIGURE 8. Fitness evolution (50 runs) of hyper-heuristics trained for 50 iterations using MAP-Elites, for different instance subsets. Three samples are shown to evidence the different behaviors achieved.



(a) Set-25-4 (b) Set-25-25 (c) Set-25-50

FIGURE 9. Median performance of hyper-heuristics ($C = 10$) trained with MAP-Elites (ME) following a reflection looping scheme. Data are shown for train and test sets. Each dot within violins represents the median performance of 50 hyper-heuristics over each of the splits for the dataset.

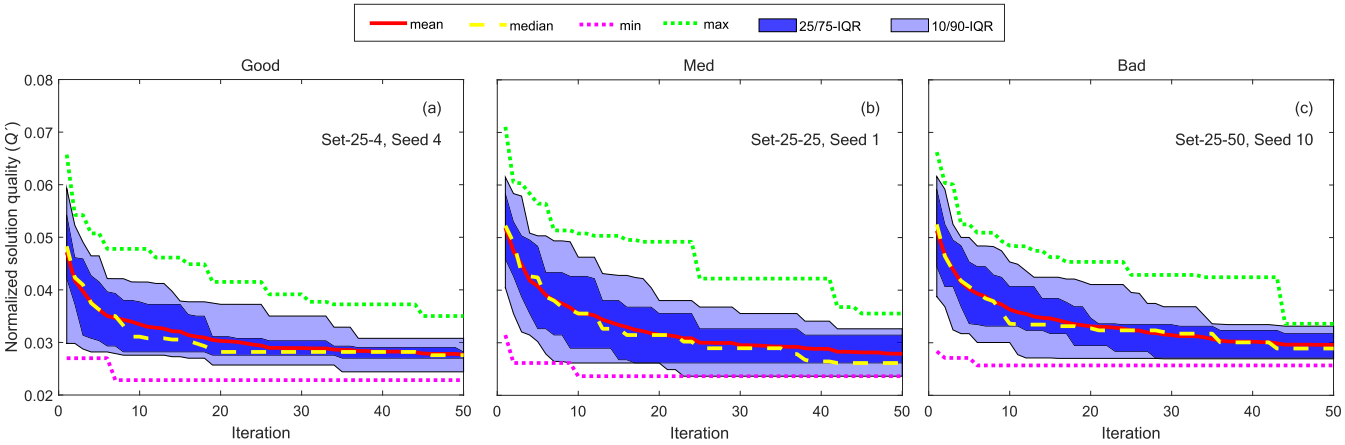


FIGURE 10. Fitness evolution (50 runs) of hyper-heuristics trained for 50 iterations using MAP-Elites with a reflection cycling scheme, for different instance subsets. Three samples are shown to evidence the different behaviors achieved.

compared the performance of short sequences. Nonetheless, we now explore such an effect in more detail, particularly for cardinality values between 5 and 20. Bear in mind that we selected such an upper limit to provide the flexibility of moving most items when solving an instance, though it is seldom required.

There are some noteworthy elements. For starters, in most scenarios, there is a similarity between cardinality values of 5 and 10, and between 15 and 20. Despite this, hyper-heuristics with a cardinality of 15 exhibit the best performance for all but one scenario in both, training and testing subsets. They are only outperformed for the hard testing subsets, where hyper-heuristics with $C = 14$ prove to be better. It implies that, as the problem becomes more challenging, it becomes harder to generalize a proper behavior for slightly larger sequences. The evidence for this rests on Figure 11c, where training with $C = 15$ provides a better performance than for $C = 14$, but which does not hold for unseen instances (Figure 11f), hinting at some kind of overfitting.

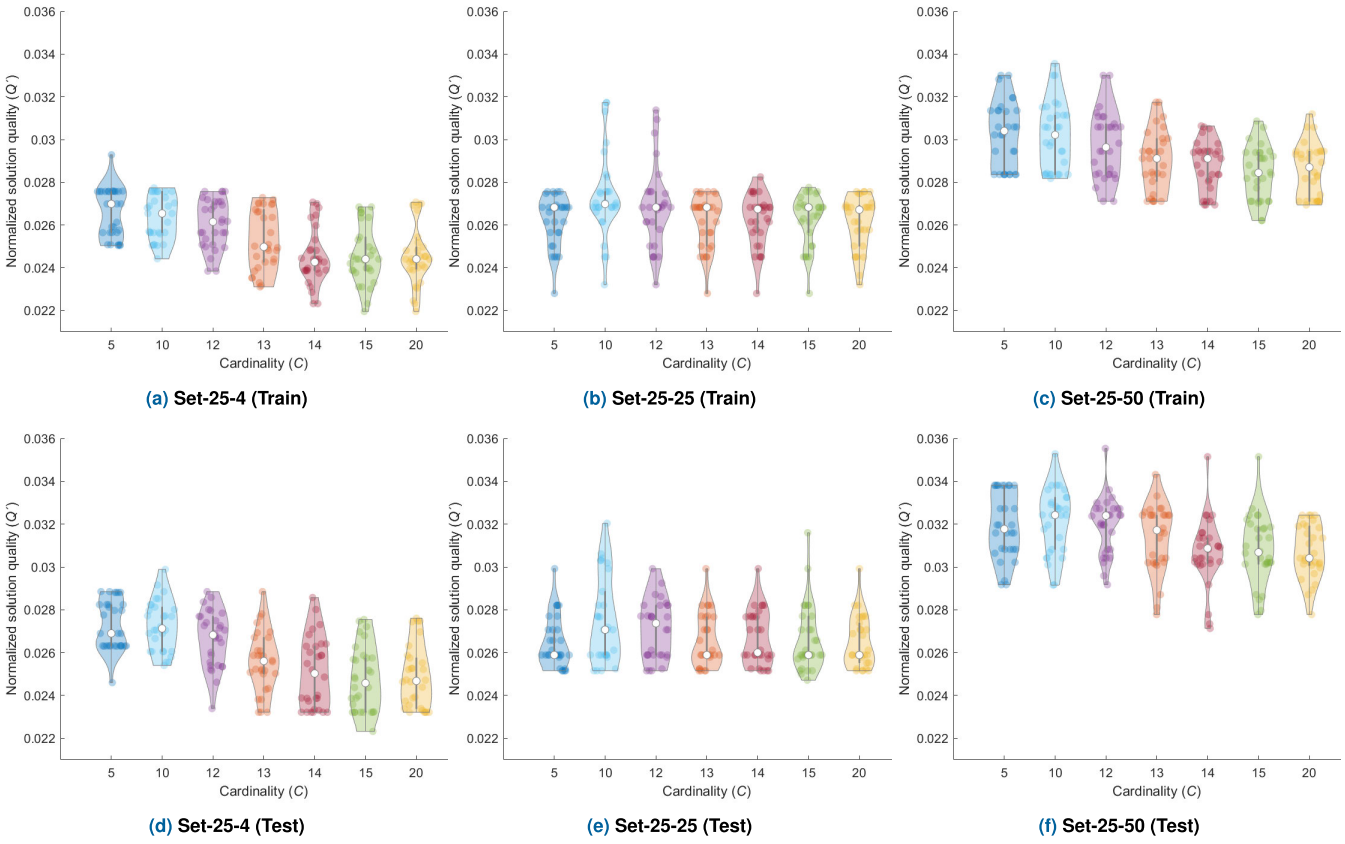
In any case, $C = 14$ and 15 are values nearly close to half the number of items within each instance. So, there may be a two-fold reason for such behavior. The first part falls to the nature of the problem: to split a set of numbers into two subsets where the total weights have the smallest possible difference. The second one rests on the nature of the instances: they are generated randomly and based on a uniform probability distribution. Hence, we can assume that instances are somewhat balanced, and thus we need to distribute about half the items. This threshold seems to apply to the three sets, and so it disregards the number of bits. In summary, in using MAP-Elites (ME), we find a generalized set of decisions for solving a set of problem instances, which may go up to the point of virtually defining one action for properly moving each item.

Let us delve deeper into the behavior of the cardinality values. Even though the median training performance for $C = 5$ and 10 in easy (Figure 11a) and hard (Figure 11c) instances is quite similar, those hyper-heuristics trained with $C = 5$ seem to generalize better to unseen instances (Figures 11d and 11f, respectively). Even so, they are unable to surpass those trained with medium cardinality values (i.e., 13–15). This strengthens the idea of not requiring heuristic sequences with cardinality values beyond half the instance length.

Such a phenomenon also sheds light on a striking pattern: there seems to be a critical value for the cardinality that represents its worst alternative. For the instances we consider, such a critical value seems to be 10–12. In this way, smaller and larger values (e.g., $C = 5$ and 15) lead to a better generalization. Of course, regard that higher cardinality values increase the computational burden as more memory is required to store the fitness map of elites. So, it becomes paramount to ponder the trade-off between performance gain and computational requirements increase.

It is also interesting to see that the behavior for the limit case is, once again, different from the others. Even so, the overall distribution from training (Figure 11b) is preserved on the testing subsets (Figure 11e). In this case, there is no relevant performance gain derived from changing the cardinality value, but using $C = 15$ seems better.

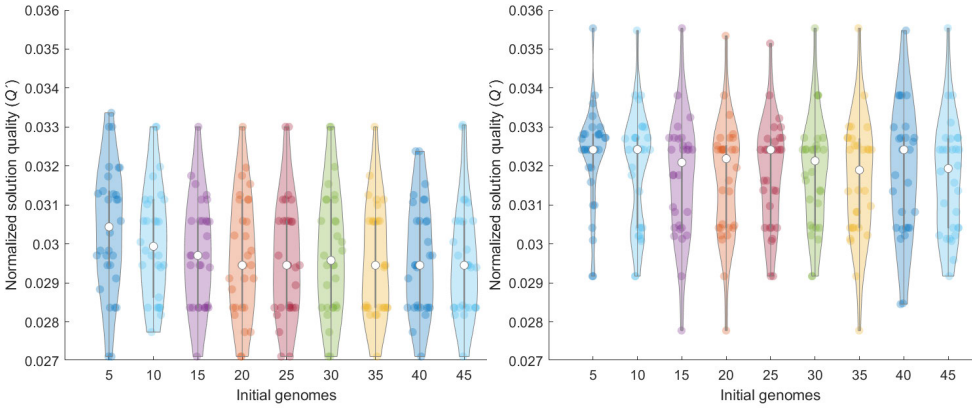
There is a behavior liaised to instance difficulty regarding the number of splits where hyper-heuristics perform better on the testing subset than on the training one. As instances become more challenging, the number of such splits diminishes, as expected. For hard instances, at most 36.67% of the splits show such an improvement, whereas for the medium and easy ones 56.67% and 53.33%, respectively, do so. It is also remarkable that, for the hard instances, the highest levels of improvements relate to $C = 5$, which then decrease monotonically. Conversely, medium instances show a stable behavior, and easy instances show an u-shape pattern while also exhibiting the highest values for $C = 5$.



(a) Set-25-4 (Train) (b) Set-25-25 (Train) (c) Set-25-50 (Train)

(d) Set-25-4 (Test) (e) Set-25-25 (Test) (f) Set-25-50 (Test)

FIGURE 11. Median performance distribution of sequence-based hyper-heuristics trained with MAP-Elites and considering different values of cardinality for each dataset. Each dot within a violin represents the median performance of 50 runs for a different instance subset.



(a) Training subset (b) Testing subset

FIGURE 12. Median performance distribution of sequence-based hyper-heuristics trained with MAP-Elites and considering different values of initial genomes for dataset Set-25-50. Each dot within a violin represents the median performance of 50 runs for a different instance subset.

3) EFFECT OF INITIAL GENOMES

Our data suggest that the number of initial genomes has little effect on the performance of hyper-heuristics. Indeed, performance for 4 and 50 bits has the same shape, while it becomes virtually unaltered for 25 bits (with some outlying values). Even so, Figure 12a shows that training with 20 or more initial genomes seemingly leads to better median performance. Moreover, for the testing set (Figure 12b) it becomes evident that hyper-heuristics trained with 15, 20, and 35 initial genomes generalize better, as indicated by the lower fitness

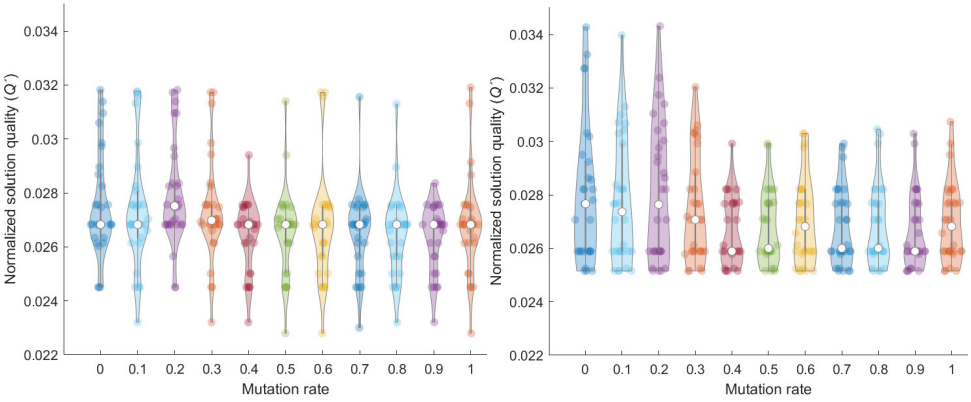


FIGURE 13. Median performance distribution of sequence-based hyper-heuristics trained with MAP-Elites and considering different values of mutation rate for dataset Set-25-25. Each dot within a violin represents the median performance of 50 runs for a different instance subset.

metric. But, using 35 initial genomes implies that the algorithm allocates 70% of the function evaluations at the start, leaving most of the process to a random search. So, we deem it better to use either 15 or 20 initial genomes.

4) EFFECT OF THE MUTATION RATE

As with the previous parameter, the effect due to mutation rate seems to be marginal. Of course, it can be because the mutation is enforced even if the mutation rate is low (see Section III-D). Consider the behaviour for 25 bits (Figure 13) as an example. In every case, a mutation rate of zero (i.e., varying a single component per iteration) has a poor median performance. As the mutation rate rises, there is a small performance improvement. Medium values, such as 0.4 and 0.5 offer a slightly better training performance (Figure 13a), while also allowing for a better generalization (Figure 13b). This strengthens the idea that variation of a single component is not disruptive enough, and so, values offering the chance of varying multiple components seem better. Nonetheless, if the value becomes too high performance again worsens as it becomes the simple random search.

5) EFFECT OF THE TRAINING RATIO

Figure 14 shows the effects of using different training ratios. For the training subset, the behavior changes depending on the kind of instance being solved. For easy instances (Figure 14a), the median performance worsens in a monotonic fashion. But, for hard instances (Figure 14c) it exhibits a multimodal behavior, with the worst median value at 30%. In the limit case (Figure 14b), such a value also peaks at 30% and from that point onward it improves.

The observed patterns can be explained by recognizing that small training ratios imply that the algorithm tailors hyper-heuristics to a handful of instances. Therefore, in some runs, it may select instances with opposing nature so that the training cannot fashion a high-performing hyper-heuristic. The opposite scenario is also feasible: a selection of instances with a similar nature. Hence, the training process yields seemingly great hyper-heuristics, even if they may suffer from overfitting. As the training ratio grows, it becomes easier to select a more diverse training set, so performance begins to stabilize.

Despite the variation inherent to the training subsets, data for testing subsets seem relatively stable. The median performance slightly improves for easy (Figure 14d) and hard (Figure 14f) instances, though it stays virtually stable for limit ones (Figure 14e). Even so, the overall behavior remains similar for training ratios of 30% and upwards. This means that, even though using more training instances leads to some improvements, the additional computational burden may not justify its usage.

Bear in mind that hyper-heuristics may use any solver at each step of the solution process. Consequently, the only way of knowing the performance over a given set of instances is to solve it. Thus, most of the computational burden associated with the training facet comes from the continuous solution of its instances. Hence, doubling the number of training instances can easily double the computational effort. Plus, going from 30% of the training instances to 90% has a huge impact while it does not even allow for a 5% reduction of the overall fitness metric. Based on this, we decide on using a training ratio of 50% for upcoming tests, seeking to preserve the improved generalization without excessively increasing the cost of training hyper-heuristics.

6) ORACLE

By this point, we have explored the effect of different parameters upon the performance of MAP-Elites. However, it is also vital to assess how such a performance relates to that of heuristics. Even so, to only consider the low-level

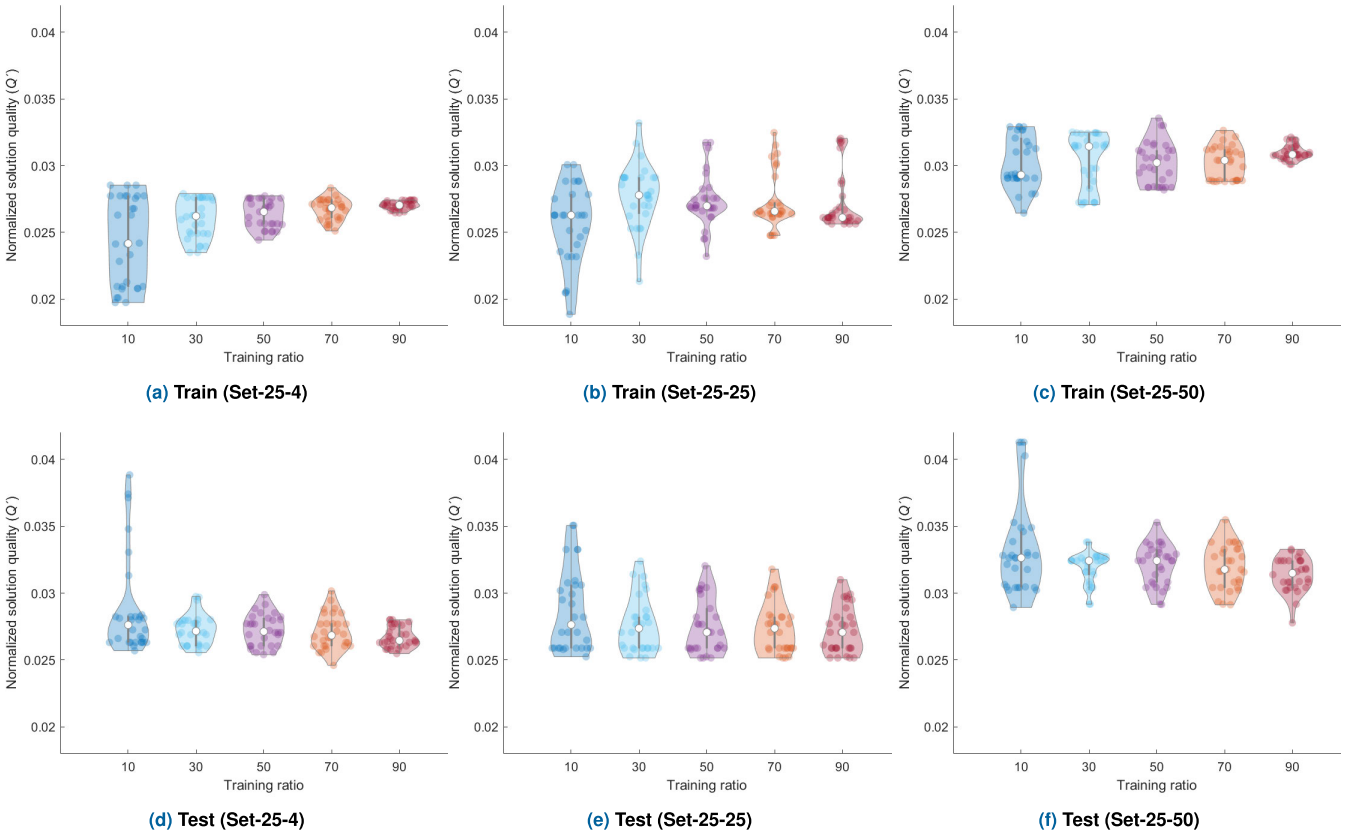


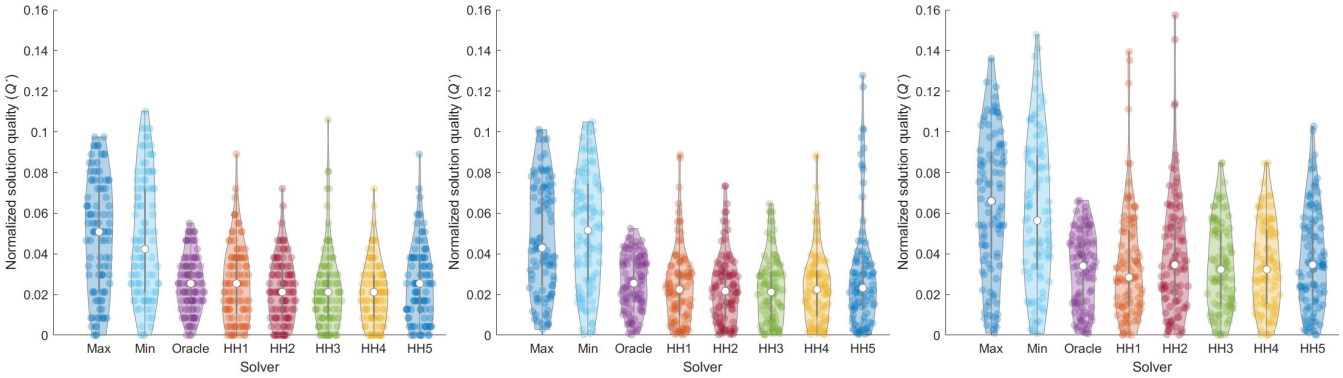
FIGURE 14. Median performance distribution of sequence-based hyper-heuristics trained with MAP-Elites and considering different training ratios (percentages) for all datasets. Each dot within a violin represents the median performance of 50 runs for a different instance subset.

heuristics used as a basis for the hyper-heuristics seems too shallow, as we already demonstrated that they could be outperformed (see Section V-A). Thus, we include an additional contestant, which corresponds to a synthetic oracle (see Section IV-B6). Take in mind that this more complex solver runs all heuristics for every instance and reports the best data. Thence, it can appropriately select among each low-level heuristic (`Max, Min`) and naturally provides better results than any of them. Although it also represents an impractical approach. Nonetheless, we deem it useful for comparison purposes.

Figure 15 shows the behavior of all solvers for the first split of all instance sets, where each dot represents the fitness of a given instance from the testing subset. As one may see, heuristics exhibit a varied performance across instances, but their union stands as a good solver. This implies that each heuristic is better at solving a different portion of the instance set. Moreover, hyper-heuristics present a valid alternative, as they not only replicate the behavior of the synthetic oracle, but also offer an improved median performance in most scenarios. For example, the oracle achieved a median quality level of 0.0279 for the easy instances, whilst the best hyper-heuristic lowered that value to 0.0175, representing a reduction of about 37%. Besides, any single hyper-heuristic outperformed the oracle in 38–53% of the corresponding instances.

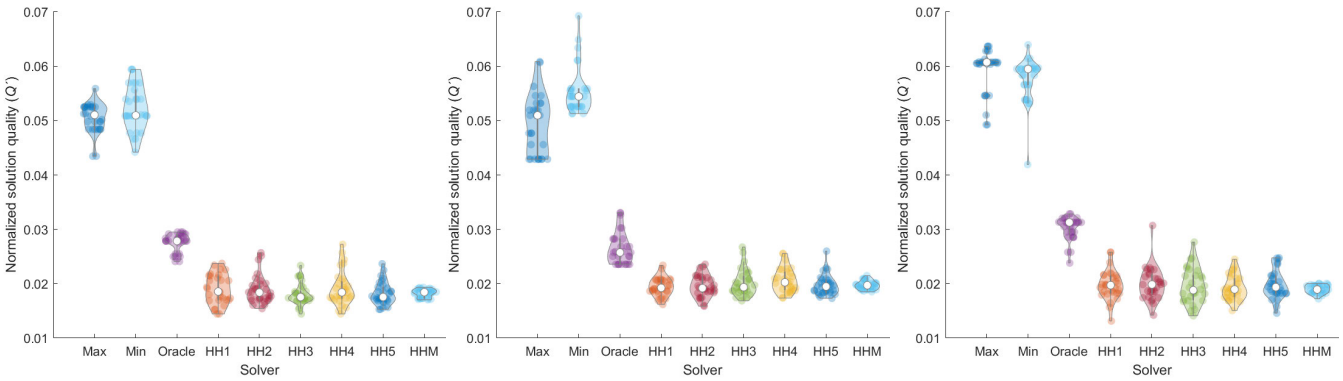
Since the behavior may change for different splits, it is also essential to analyze the global landscape. So, Figure 16 condenses data for all 30 splits. In each split, we generated new hyper-heuristics that adapt to its particular training data. So, each dot within a violin represents the average quality level for the testing subset of a given seed. As one may see, low-level heuristics perform poorly when used by themselves. In contrast, the oracle shows a meaningful improvement. This means that datasets are varied and that it is paramount to select among available heuristics properly.

It is noteworthy that hyper-heuristics match the oracle performance level, and for some cases, they even surpass it. The distribution of performance levels for all datasets is clearly better than for the oracle, except for some outlying instances. Besides, hyper-heuristics have a somewhat stable performance across different runs, especially in terms of median performance. This is evident by the small range of the last violin, where we condense the median performance of all 50 runs across each split. Even in the worst-case scenario (an instance in the second run of Set-25-50), the hyper-heuristic performed better than all heuristics when solving



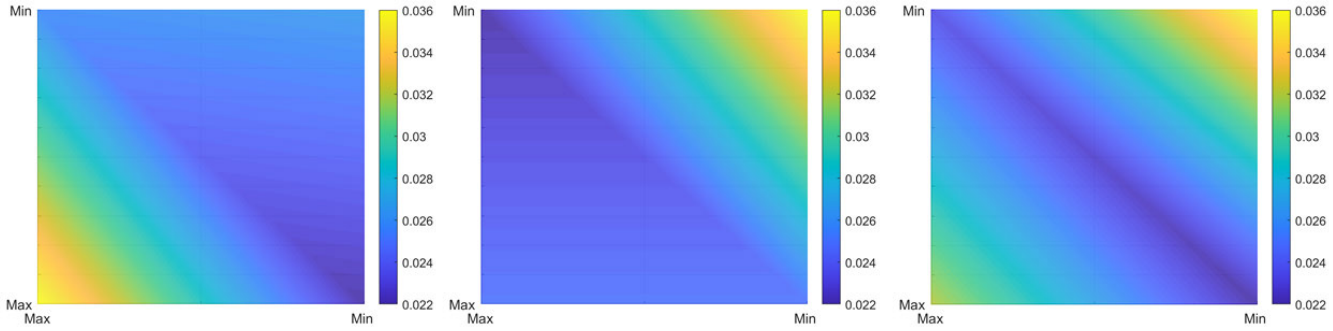
(a) Set-25-4 (b) Set-25-25 (c) Set-25-50

FIGURE 15. Performance achieved by all solvers over each instance in the first split (testing subset) of all datasets.



(a) Set-25-4 (b) Set-25-25 (c) Set-25-50

FIGURE 16. Performance achieved by all solvers for each split of all datasets (testing subsets). HHM shows the median performance of 50 hyper-heuristics trained on their corresponding splits.



(a) Set-25-4 (b) Set-25-25 (c) Set-25-50

FIGURE 17. Hyper-heuristic performance of feasible designs illuminated by MAP-Elites after 100 function evaluations. Each plot condenses data for 15 dimensions observed from the first two.

their best instance. What is more, there is always a minimum of one hyper-heuristic that outperformed the oracle for all datasets. These results imply that hyper-heuristics clearly outperform single low-level heuristics and even their ideal selector.

Since the idea of MAP-Elites is to provide information about the performance of different "design alternatives," we now analyze some of the generated grids. Figure 17 displays the performance map for one split (training subset) of all datasets from the first two decisions perspective. Notice that

each dimension only has two points, which correspond to the available heuristics at these steps. Nonetheless, the total number of dimensions equals the cardinality, so each map shows the information for 15 dimensions. We collapsed the remaining dimensions by analyzing the best possible scenario. Also, observe that we increase the number of function evaluations to 100 for these plots to achieve a better-illuminated map.

These maps reflect that the highest-performing sequences vary across sets. Easy (Set-25-4, Figure 17a) and limit (Set-25-25, Figure 17b) instances exhibit a clearly opposing behavior, where the highest-performing solution starts with the opposite sequence ({`Min,Max`} and {`Max,Min`}, respectively) and where the worst-performing solution starts by repeating opposing heuristics (`Min` and `Max`, respectively). In contrast, hard instances (Set-25-50, Figure 17c) show a more balanced performance, where both combinations are right, and both standalone heuristics perform poorly. In any case, the first two steps are paramount for the performance of hyper-heuristics. Hence, a poor initial choice can be critical for the whole sequence. Even though performance changes from one kind of instance to the other, it is always better to mix heuristics at the first two decisions, even if one cannot determine the best combination.

C. ADVANCED TESTS

Now that we have verified the effect of diverse algorithm parameters, we discuss the resulting data from more complex tests to validate its capabilities.

1) EFFECT OF INCREASING THE POOL OF HEURISTICS

Figure 18 presents the performance of ME on all splits of every set when using the pool of five heuristics (see Section III-C): `Max`, `Min`, `2-Max`, `2-Min`, and `Median`. Each data point is given by the best performance achieved by MAP-Elites in the training subsets after 50 function evaluations. Moreover, each violin contains 50 points, representing the average performance of each run for the corresponding split.

There are notable differences in the behavior, compared to the case using two heuristics (see Figure 6). For starters, performance across runs follows a different distribution. In the current case, each shape is softer, whilst in the previous one (two heuristics), it exhibited condensed regions with several extreme values.

A second difference is that stability improves. In the case of two heuristics, the median performance oscillates in spans of 0.004, 0.011, and 0.006, for instances with 4, 25, and 50 bits, respectively. By increasing the heuristic pool to five, such ranges diminish to 0.003, 0.004, and 0.003. Not only this represents a reduction between 25–64%, but it also hints at a lower sensitivity to instance difficulty. Moreover, actual median performance levels are better for all but one of the tested subsets. The only exception is one split of instances with 25 bits, but even in this case, both median performances are close (a difference of about 0.001).

In a general sense, hyper-heuristics that draw from a larger pool of heuristics exhibit better performance. This becomes evident from comparing the best and worst runs of hyper-heuristics choosing among two and five heuristics. Figure 6 showed that bad runs yielded a normalized quality of about 0.039, 0.043, and 0.043, for 4, 25, and 50 bits, respectively. Meanwhile, Figure 18 depicts that analog values sit about the 0.032, 0.037, and 0.037 marks. Similarly, best runs when considering five heuristics yield values unfathomable for the case of two heuristics.

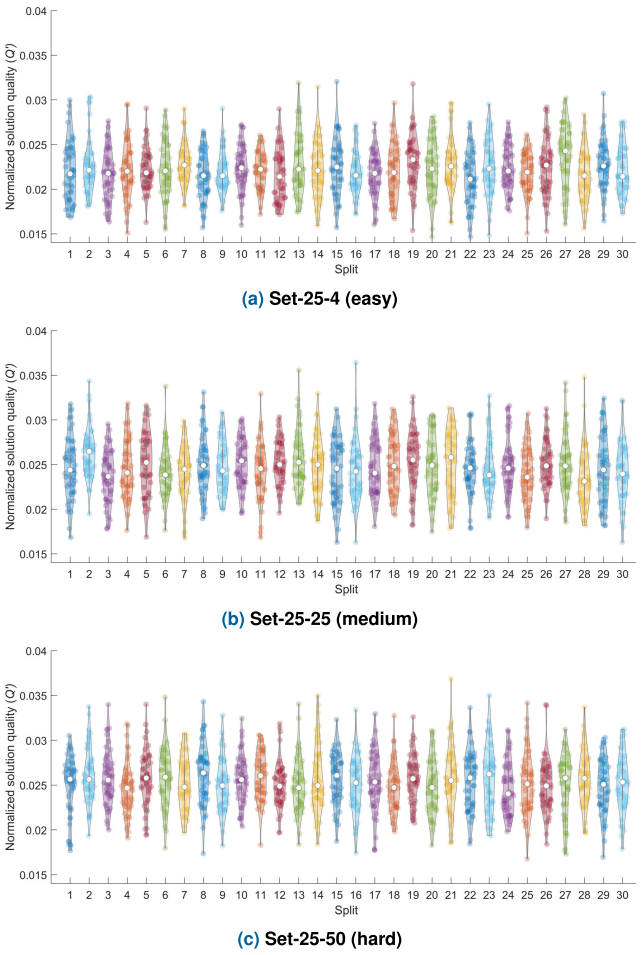
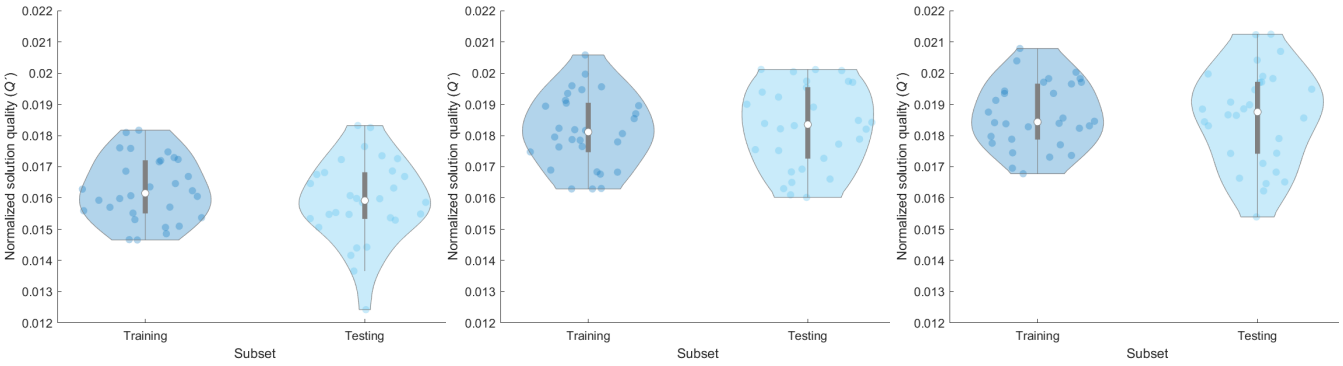


FIGURE 18. Performance achieved by sequence-based hyper-heuristics ($C = 10$) trained with MAP-Elites over different training splits of the datasets, using five heuristics. Each dot represents the best fitness achieved after 50 function evaluations. Each violin contains 50 points, representing all runs for that split.

Figure 19 presents the minimum average fitness achieved across the 50 runs of each split in both training and testing subsets. It is impressive to see that the algorithm seems to generalize quite well, up to the point that it always achieves better results in the testing subsets than in the training ones. This illustrates another benefit of using the larger heuristic pool since, in the previous case, data for testing subsets were slightly worse than for training. In fact, there is an improvement in the median value of this metric for all training



(a) Set-25-4 (b) Set-25-25 (c) Set-25-50

FIGURE 19. Performance achieved by sequence-based hyper-heuristics ($C = 10$) trained with MAP-Elites over training and testing subsets of Set-25-4, Set-25-25, and Set-25-50 using five heuristics. Each violin contains 30 dots, where each one represents the best performance across 50 runs of a different subset.

and testing scenarios. It represents fitness reductions between 27% and 44% w.r.t. the values achieved when working with a pool of two heuristics. Moreover, such reductions represent between 184% and 212% of the range achieved with the reduced pool.

Table 3 shows the p-values resulting from the Wilcoxon statistical test comparing the data for both pool sizes. We run such tests for both subsets (training and testing) and focus on different parameters (best, worst, and median performance across runs). So, these values prove that increasing the pool size leads to different performance distributions, at a significance level of 0.05. Besides, for most cases, this conclusion holds even at a significance level of 0.01. In particular, the best p-values are associated with the best runs, indicating that it could be used as a metric for comparing performance, and which relates to improvements in the best-case scenarios that are larger than for the others.

TABLE 3. p-values associated to a Wilcoxon statistical test comparing the results of experiments using pools of two and five heuristics. All tests reveal statistically significant differences at $\alpha = 0.05$.

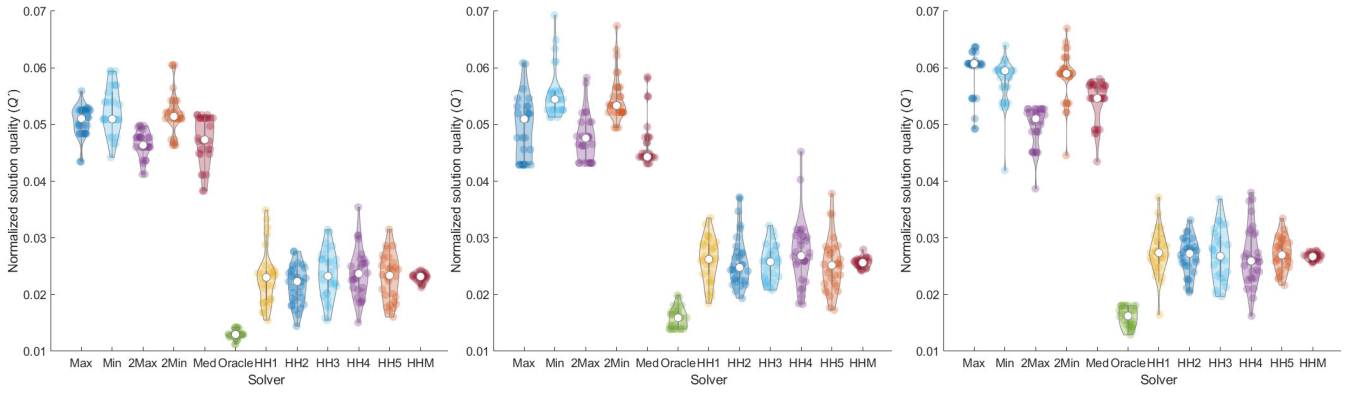
Parameter	Subset	4 bits ($\times 10^{-11}$)	25 bits ($\times 10^{-11}$)	50 bits ($\times 10^{-11}$)
Median	Train	2.88	2.13×10^{3}	2.90
	Test	2.97	1.03×10^{7}	2.99
Best	Train	2.86	2.75	2.83
	Test	2.75	2.75	2.83
Worst	Train	97.2	128	203
	Test	1.44×10^{9}	2.80×10^{8}	8.72×10^{4}

Let us now compare how this improved model fares against an improved oracle that selects among the five heuristics (Figure 20). Such an oracle also shows a clear improvement derived from the extra heuristics (cf. Section V-B6). This means that the newly added heuristics are indeed better at solving some of the instances. Otherwise, the oracle would not have selected them, and its performance would have remained identical. Moreover, even though hyper-heuristics also improved significantly, they did not do it as much as the oracle. So, hyper-heuristics can no longer beat the oracle. This is confirmed by the last violin, which once again condenses the median performance of the 50 hyper-heuristics. Nonetheless, their behavior remains relatively better than that of standalone heuristics. Moreover, they share some common regions with the oracle, meaning that they may have replicated it for some instances. Even so, such regions become smaller as the problem grows in difficulty.

At this point, it becomes necessary to remind the reader about the training process of hyper-heuristics. For all tests, we considered a maximum of 50 function evaluations, including those required for initialization. This means that we trained hyper-heuristics with similar computational resources when accessing two and five heuristic pools. But, the complexity of each problem is far from similar. Remember that ME illuminates the search domain, and hence seeks to explore different solutions to a problem. Because of this, it keeps track of a map of solutions. In our case, each dimension of such a map is given by an element of the hyper-heuristic, and its domain equals the number of available heuristics. When considering the pool of two heuristics, ME seeks to illuminate a space of 32768 ($= 2^{15}$) combinations. But, for the other one, such a space grows to 5^{10}, nearly representing a 300-fold increase. Taking this into account, it is commendable that ME procures hyper-heuristics with such a performance level as it only explores a tiny fraction of the search domain (about 0.0005%).

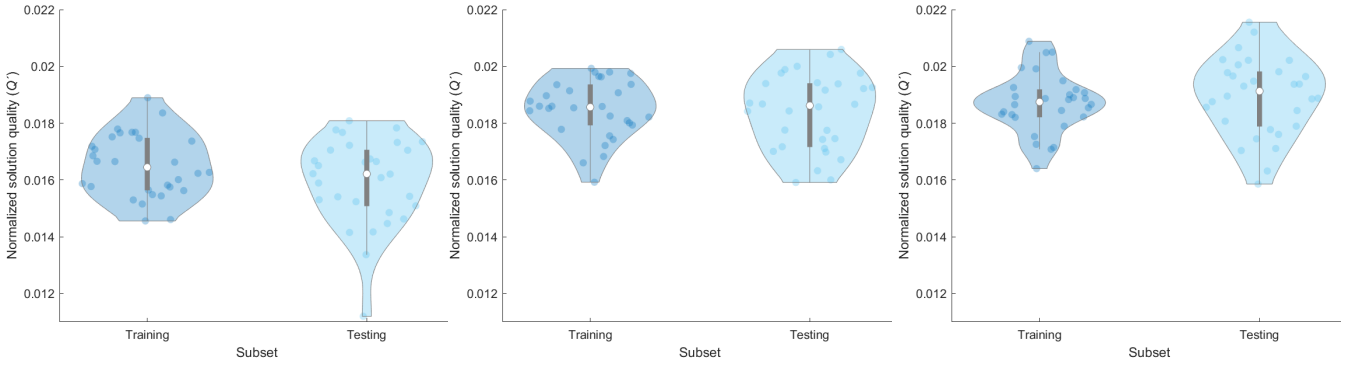
2) EFFECT OF MUTATION OPERATORS

As a complementary test, we now change the mutation scheme while preserving the increased pool of heuristics. For this, we implement five kinds of mutation operators (see Section IV-C2). The resulting data is quite similar to that from the previous strategy (see Section V-C1), but we omit detailed plots for the sake of brevity. Instead, it should suffice



(a) Set-25-4 (b) Set-25-25 (c) Set-25-50

FIGURE 20. Performance achieved by all solvers (extended pool size) for each split of all datasets (testing subsets). HHM shows the median performance of 50 hyper-heuristics trained on their corresponding splits.



(a) Set-25-4 (b) Set-25-25 (c) Set-25-50

FIGURE 21. Performance achieved by sequence-based hyper-heuristics ($C = 10$) trained with MAP-Elites (ME) over training and testing subsets of Set-25-4, Set-25-25, and Set-25-50 using five heuristics and five mutation operators. Each violin contains 30 dots, where each one represents the best performance across 50 runs of a different subset.

to say that median values and ranges for all runs of each of the 30 splits are quite similar for both mutation schemes.

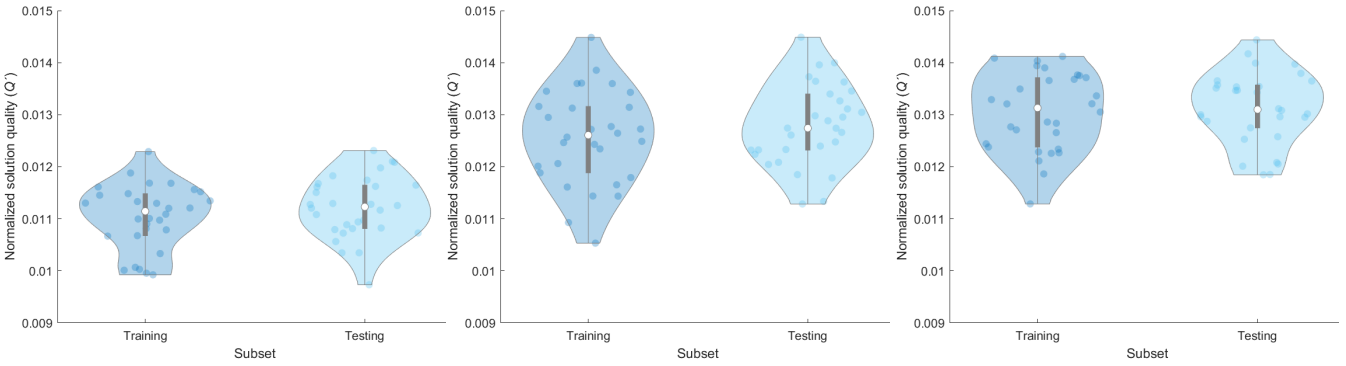
We also present the minimum average fitness variation for each split (Figure 21). We choose this metric as it stands as the one where differences are more evident. Also, bear in mind that these values represent the best of 50 runs for each split. It seems as if using this mutation scheme is counter-productive for the proposed approach, as there is virtually no improvement from training to testing subsets. Au contraire, the previous scheme allowed for improvements in all scenarios (Figure 19). Despite this, both approaches provide similar performance levels for the instances, which are not statistically significant (Table 4).

TABLE 4. *p*-values associated to a Wilcoxon statistical test comparing the results of using different mutation schemes. Highlighted values indicate the cases where performance differences are not statistically significant.

Parameter	Subset	4 bits ($\times 10^{-3}$)	25 bits ($\times 10^{-3}$)	50 bits ($\times 10^{-3}$)
Median	Train	0.141	0.016	8.00
	Test	0.104	0.526	8.00
Best	Train	**994.0**	**158.0**	**58.40**
	Test	**367.0**	**344.0**	**67.90**
Worst	Train	0.104	0.064	2.90
	Test	**53.70**	8.30	24.20

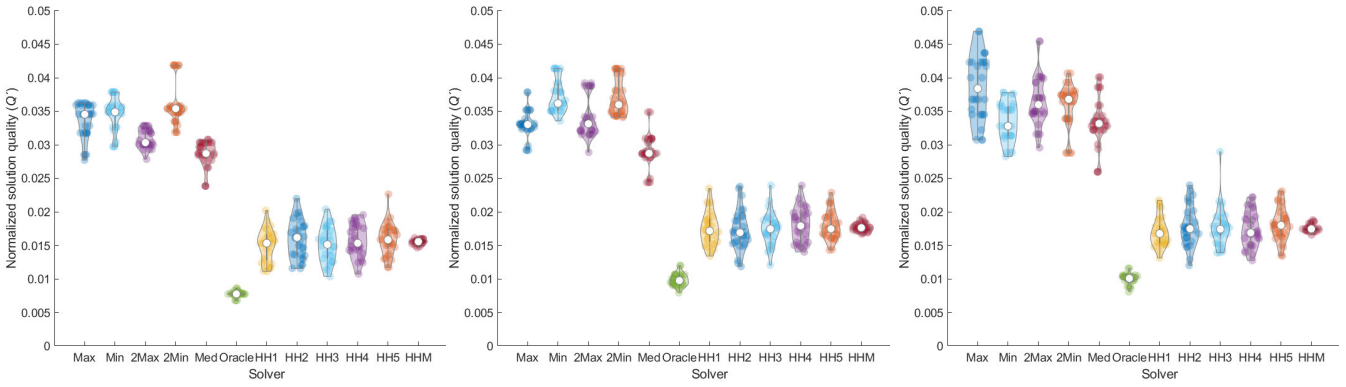
3) EFFECT OF SETS WITH MORE ELEMENTS

Increasing the number of elements within instances, i.e., using larger instances, does not hinder the proposed model performance. Instead, the behavior from training to testing holds whilst the overall performance improves. This is somewhat expected as larger instances are supposed to be easier than shorter ones (for the same number of bits per element). Figure 22 shows the distribution of the best runs (out of 50) for all training and testing subsets. The improvement w.r.t. datasets with shorter instances (Figure 19 and Figure 21) is clear. There is a reduction between 15% and 30% for this



(a) Set-40-4 (b) Set-40-25 (c) Set-40-50

FIGURE 22. Performance achieved by sequence-based hyper-heuristics ($C = 10$) trained with MAP-Elites (ME) over training and testing subsets of Set-40-4, Set-40-25, and Set-40-50 using five heuristics and five mutation operators. Each violin contains 30 dots, where each one represents the best performance across 50 runs of a different subset.



(a) Set-40-4 (b) Set-40-25 (c) Set-40-50

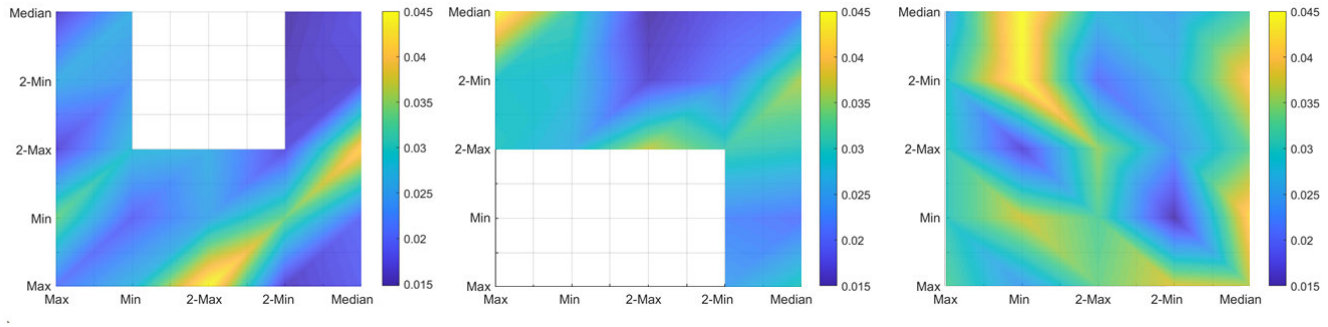
FIGURE 23. Performance achieved by all solvers (extended pool size) for each split of all large datasets (testing subsets), considering the set of five mutation operators. HHM shows the median performance of 50 hyper-heuristics trained on their corresponding splits.

metric in all training and testing subsets. Moreover, there are some changes in performance distribution shapes, although they are more evident for the training than testing subsets. In particular, the performance for these instances seems more evenly distributed along with the range of achieved values.

The instances shown in Figure 21c represent an example where training performance seems to follow a normal distribution with a mean of 0.0187 and a standard deviation of about 0.0011. But with larger instances (Figure 22c), the distribution becomes almost uniform for most of the range. Even if this seems like a drawback (less stable performance), it is quite the opposite. Training performance in shorter instances spans for about 0.0045 units, whilst for larger instances, it diminishes to about 0.0030. Considering that median values improve for larger instances, one may conclude that performance is better for these kinds of instances. A statistical test between both performance sets reveals a p-value of 3.02×10^{-11}, indicating a significant difference and reinforcing such a conclusion.

Let us now compare how the model fares against low-level heuristics and their oracle (Figure 23). The overall behavior is similar to that shown for shorter instances (Figure 20). However, all solvers improved considerably; i.e., approximately 34.19% of the previous fitness metric. But, the oracle remains undefeated. Nonetheless, hyper-heuristics are pretty better than standalone heuristics. The performance metric for the median hyper-heuristic (last violin of each subplot) is, on average, 44.34% smaller than the best median performance achieved by any standalone heuristic for any difficulty. Also, once again, the model seems quite stable. The performance of the median hyper-heuristic oscillates across the different splits with a standard deviation of about 2.44×10^{-17} units. This is noteworthy as the oracle does so with an approximate value of 0.0078. So, our proposed approach leads to models less sensitive to instance length.

Likewise, we analyze some sample maps achieved with our proposal (Figure 24). Bear in mind that we still run these maps for 100 function evaluations, although the search space is way bigger (5^{10} vs. 2^{15}). So, in some cases, it becomes



(a) Set-40-4 (b) Set-40-25 (c) Set-40-50

FIGURE 24. Hyper-heuristic performance of feasible designs illuminated by MAP-Elites after 100 function evaluations. Each plot condenses data for 10 dimensions observed from the first two. Blank spaces represent regions where one at least one vertex did not have performance data.

impossible to gather enough data for a complete map (we are only evaluating 0.001% of the whole domain). Still, it is noteworthy that some insights appear.

One of such insights is that the `Median` heuristic should be avoided as the first element in the sequence when solving hard instances, as it leads to poorly performing hyper-heuristics. However, it should only be avoided for easy instances if the second step corresponds to the `2-Max` heuristic. Similarly, leading the solution process with heuristic `2-Min` in easy and hard instances works best. However, it must be followed by heuristics `Median` or `Min`, depending on the difficulty level of the instance. In any case, using the same heuristic for the first couple steps is a bad idea (Figure 24). Although the hindrance is not as effective as for smaller instances (cf. Figure 17), it remains paramount to select a proper combination of initial steps, as Figure 24c shows.

VI. CONCLUSION AND FUTURE WORK

In this work, we presented a constructive, sequence-based, selection hyper-heuristic model powered by MAP-Elites (ME), and assessed its feasibility. The reason: traditional HHs fall short when providing information about their sensitivity. Our approach seeks to find a proper combination of heuristics for tackling combinatorial optimization problems, while simultaneously providing information about the performance level that can be expected if some changes are incorporated into the model. To test the proposed approach, we tackled 1500 instances of the Balanced Partition problem. Our testing included randomly generated instances with three difficulty levels: easy, hard, and the limit case.

Data reveal that training with ME yields better results than when one randomly selects heuristics. This is remarkable as the model does not require features to analyze problem states. The resulting hyper-heuristics are competitive. With a reduced pool of heuristics, they always outperform the oracle. With an extended pool of heuristics, performance worsens, and although they replicate the oracle for some of instances, they no longer outperform it. We believe this is due to the explosion of the search domain, as we use the same amount of computational resources for both kinds of tests. But the search domain for the reduced pool was of 2^{15} combinations, whereas that of the extended pool surged to 5^{10}. So, allowing for more iterations in the training phase should improve the performance levels of the generated hyper-heuristics. Nonetheless, bad performance levels are still better than those of any heuristic when acting in a standalone fashion.

We also found that some parameters of our proposed approach have a high impact on performance, while others slightly affect it. Our data revealed that the cycling scheme (restart or reflection), the cardinality (length) of a hyper-heuristic, the training ratio, the heuristic pool size, and the instance size, are crucial. For example, extending easy instances from 25 to 40 elements enhanced the median hyper-heuristic performance by 33.16%. Conversely, the number of initial genomes and the mutation rate did not affect much. Increasing initial genomes from 5 to 20 while dealing with hard instances only hindered the median training performance by 3.25%. Even a different mutation scheme, which we considered to be more disruptive, had a modest effect. Perhaps an even more robust scheme is required, and we should explore this in future work.

Illuminating the space of ''design alternatives'' also led to some insights. First, there is no single heuristic combination (i.e., a sequence) that performed best for all cases. Second, the first two decisions seemed paramount for hyper-heuristic performance, even on larger sets. Even so, mixed sequences were better than using the same heuristic twice. Although for larger sets the effect diminished, it remained relevant. Finally, this best initial combination of heuristics appeared to change depending on the difficulty of the instance.

As it is only natural, our work has some drawbacks that should be addressed. First and foremost: memory requirements. MAP-Elites requires storing the whole solution grid, which in our case consisted of one dimension per cardinality level, with a search domain equal to the number of heuristics (per dimension). Moreover, we used Matlab for coding the model, and a modest laptop for running the experiments. Under these conditions, it was impossible to train HHs of cardinality 15 and 5 heuristics. We somewhat expected this issue, as this implies storing a grid with 5^{15} positions. Nonetheless,

one may lessen its effect by migrating to more efficient programming languages or data structures.

Another drawback relates to the solution model and the test bench. For this work, we restricted ourselves to a sequence-based selection hyper-heuristic model, though other models exist. Notwithstanding, including more models would extend the manuscript beyond a reasonable length. Similarly, we only considered Balanced Partition for the sake of brevity.

We believe that these drawbacks are more of an opportunity than an issue. So far, we have shown that the proposed model works, though it remains open to improvements. For example, right now the cycling scheme is fixed. Hence, it shall prove interesting to analyze how performance changes when adding the cycling scheme as a variable to learn whether to use reflection, restart, or other schemes.

Similarly, one may easily extend the model to include more disruptive mutation operators that change the cardinality by adding or removing steps. It shall also prove interesting to analyze the behavior of the model for instances of varying size and different nature. Even better, one may use an instance generator for tailoring instances to each heuristic and thus create a more robust training dataset.

Anther path includes the development of a rule-based hyper-heuristic. In doing so, one may use ME to obtain information about the effect of different sets of rules. However, finding the necessary features of each domain and linking rules to a grid for ME represent drawbacks to overcome. We believe that the following step must test the proposed model with more complex combinatorial optimization problems, such as Job-Shop Scheduling or Knapsack. This would allow for a more global assessment of its performance.

Finally, it is important to address that although this work focuses on domain-specific hyper-heuristics, there are other approaches. For example, the HyFlex framework seeks to create cross-domain hyper-heuristics following a perturbative approach [69]. So, one can also improve upon this work by using ME to generate such kinds of hyper-heuristics.

REFERENCES

[1] J.-J. Han, Z. Wang, S. Gong, T. Miao, and L. T. Yang, "Resource-aware scheduling for dependable multicore real-time systems: Utilization bound and partitioning algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2806–2819, Dec. 2019.

[2] G. P. Georgiadis, B. M. Pampín, D. A. Cabo, and M. C. Georgiadis, "Optimal production scheduling of food process industries," *Comput. Chem. Eng.*, vol. 134, Mar. 2020, Art. no. 106682.

[3] M. Sanchez, J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, H. Ceballos, H. Terashima-Marin, and I. Amaya, "A systematic review of hyper-heuristics on combinatorial optimization problems," *IEEE Access*, vol. 8, pp. 128068–128095, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9139914/

[4] Y. Yao, Z. Peng, and B. Xiao, "Parallel hyper-heuristic algorithm for multi-objective route planning in a smart city," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10307–10318, Nov. 2018.

[5] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. Burke, "HyFlex: A benchmark framework for cross-domain heuristic search," in *Proc. 12th Eur. Conf. Evol. Comput. Combinat. Optim.*, Berlin, Germany, 2012, pp. 136–147. [Online]. Available: http://arxiv.org/abs/1107.5462

[6] C. Ansótegui, J. Gabàs, Y. Malitsky, and M. Sellmann, "MaxSAT by improved instance-specific algorithm configuration," *Artif. Intell.*, vol. 235, pp. 26–39, Jun. 2016. http://linkinghub.elsevier.com/retrieve/pii/S0004370215001824, doi: 10.1016/j.artint.2015.12.006.

[7] Y. Malitsky and M. Sellmann, "Instance-specific algorithm configuration as a method for non-model-based portfolio generation," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Lecture Notes in Computer Science), N. Beldiceanu, N. Jussien, and E. Pinson, Eds. Berlin, Germany: Springer, 2012, pp. 244–259, doi: 10.1007/978-3-642-29828-8_16.

[8] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, nos. 2–3, pp. 95–99, 1988.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[10] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inf. Sci.*, vol. 237, pp. 82–117, Jul. 2013.

[11] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Comput. Ind. Eng.*, vol. 137, Nov. 2019, Art. no. 106040, doi: 10.1016/j.cie.2019.106040.

[12] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: A comprehensive survey," *Artif. Intell. Rev.*, vol. 52, no. 4, pp. 2191–2233, Dec. 2019, doi: 10.1007/s10462-017-9605-z.

[13] A. Vela, J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, and I. Amaya, "Tailoring job shop scheduling problem instances through unified particle swarm optimization," *IEEE Access*, vol. 9, pp. 66891–66914, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9418993/

[14] J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, I. Amaya, Y. Shi, H. Terashima-Marín, and N. Pillay, "Towards a generalised metaheuristic model for continuous optimisation problems," *Mathematics*, vol. 8, no. 11, p. 2046, Nov. 2020. [Online]. Available: https://www.mdpi.com/2227-7390/8/11/2046

[15] N. Pillay and R. Qu, *Hyper-Heuristics: Theory and Applications* (Natural Computing Series). Cham, Switzerland: Springer, 2018, doi: 10.1007/978-3-319-96514-7.

[16] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *Eur. J. Oper. Res.*, vol. 285, no. 2, pp. 405–428, Sep. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0377221719306526, doi: 10.1016/j.ejor.2019.07.073.

[17] I. Amaya, J. C. Ortiz-Bayliss, A. Rosales-Pérez, A. E. Gutiérrez-Rodríguez, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. C. Coello, "Enhancing selection hyper-heuristics via feature transformations," *IEEE Comput. Intell. Mag.*, vol. 13, no. 2, pp. 30–41, May 2018.

[18] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, *A Classification of Hyper-Heuristic Approaches: Revisited* (International Series in Operations Research & Management Science), vol. 272. Cham, Switzerland: Springer, 2019, pp. 453–477. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-91086-4_14

[19] P. Cowling, G. Kendall, and E. Soubeiga, "Adaptively parameterised hyperheuristics for sales summit scheduling," in *Proc. Sel. 4th Metaheuristics Int. Conf.*, 2001, pp. 1–22.

[20] F. Garza-Santisteban, R. Sanchez-Pamanes, L. A. Puente-Rodriguez, I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, and H. Terashima-Marin, "A simulated annealing hyper-heuristic for job shop scheduling problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 57–64. [Online]. Available: https://ieeexplore.ieee.org/document/8790296/

[21] C. Yu, P. Andreotti, and Q. Semeraro, "Multi-objective scheduling in hybrid flow shop: Evolutionary algorithms using multi-decoding framework," *Comput. Ind. Eng.*, vol. 147, Sep. 2020, Art. no. 106570, doi: 10.1016/j.cie.2020.106570.

[22] C.-C. Wu, D. Bai, J.-H. Chen, W.-C. Lin, L. Xing, J.-C. Lin, and S.-R. Cheng, "Several variants of simulated annealing hyper-heuristic for a single-machine scheduling with two-scenario-based dependent processing times," *Swarm Evol. Comput.*, vol. 60, Feb. 2021, Art. no. 100765, doi: 10.1016/j.swevo.2020.100765.

[23] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," 2015, *arXiv:1504.04909*. [Online]. Available: http://arxiv.org/abs/1504.04909

[24] N. Urquhart and E. Hart, *Optimisation and Illumination of a Real-World Workforce Scheduling and Routing Application (WSRP) Via MapElites* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11101. Cham, Switzerland: Springer, 2018, pp. 488–499. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-99253-2_39#citeas

[25] M. Maashi, E. Özcan, and G. Kendall, "A multi-objective hyper-heuristic based on choice function," *Expert Syst. Appl.*, vol. 41, no. 9, pp. 4475–4493, Jul. 2014, doi: 10.1016/j.eswa.2013.12.050.
[26] M. Maashi, G. Kendall, and E. Özcan, "Choice function based hyper-heuristics for multi-objective optimization," *Appl. Soft Comput.*, vol. 28, pp. 312–326, Mar. 2015, doi: 10.1016/j.asoc.2014.12.012.
[27] M. S. Maashi, "Multi-objective hyper-heuristics," in *Heuristics and Hyper-Heuristics: Principles and Applications*, vol. 32. Rijeka, Croatia: InTech, Aug. 2017, pp. 137–144. [Online]. Available: http://www.intechopen.com/books/trends-in-telecommunications-technologies/gps-total-electron-content-tec-prediction-at-ionosphere-layer-over-the-equatorial-region0AInTec0A and http://www.asociatiamhc.ro/wp-content/uploads/2013/11/Guide-to-Hydropower.pdf
[28] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.
[29] S. Fioravanzo and G. Iacca, "Evaluating MAP-elites on constrained optimization problems," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2019, pp. 253–254.
[30] B. Hayes, "Computing science: The easiest hard problem," *Amer. Scientist*, vol. 90, no. 2, pp. 113–117, 2002.
[31] C. Wu, E. Kamar, and E. Horvitz, "Clustering for set partitioning with a case study in ridesharing," in *Proc. IEEE 19th Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2016, pp. 1384–1388.
[32] S. Soltan, M. Yannakakis, and G. Zussman, "Doubly balanced connected graph partitioning," in *Proc. 28th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2017, pp. 1939–1950.
[33] W. Fan, M. Liu, C. Tian, R. Xu, and J. Zhou, "Incrementalization of graph partitioning algorithms," *Proc. VLDB Endowment*, vol. 13, no. 8, pp. 1261–1274, 2020.
[34] D. Solow, J. Ning, J. Zhu, and Y. Cai, "Improved heuristics for finding balanced teams," *IISE Trans.*, vol. 52, no. 12, pp. 1312–1323, Dec. 2020.
[35] A. Moreno, P. Munari, and D. Alem, "Decomposition-based algorithms for the crew scheduling and routing problem in road restoration," *Comput. Oper. Res.*, vol. 119, Jul. 2020, Art. no. 104935, doi: 10.1016/j.cor.2020.104935.
[36] A. Tafreshian and N. Masoud, "Trip-based graph partitioning in dynamic ridesharing," *Transp. Res. C, Emerg. Technol.*, vol. 114, pp. 532–553, May 2020, doi: 10.1016/j.trc.2020.02.008.
[37] V. Buchhold, D. Delling, D. Schieferdecker, and M. Wegner, "Fast and stable repartitioning of road networks," in *Proc. 18th Int. Symp. Exp. Algorithms*, no. 26, 2020, pp. 1–15.
[38] W. No, J. Choi, S. Park, and D. Lee, "Balancing hazard exposure and walking distance in evacuation route planning during earthquake disasters," *ISPRS Int. J. Geo-Inf.*, vol. 9, no. 7, p. 432, Jul. 2020.
[39] C. Billing, F. Jaehn, and T. Wensing, "Fair task allocation problem," *Ann. Oper. Res.*, vol. 284, no. 1, pp. 131–146, Jan. 2020, doi: 10.1007/s10479-018-3052-3.
[40] Z.-W. Chen, H. Lei, M.-L. Yang, Y. Liao, and J.-L. Yu, "Improved task and resource partitioning under the resource-oriented partitioned scheduling," *J. Comput. Sci. Technol.*, vol. 34, no. 4, pp. 839–853, Jul. 2019.
[41] J. Wang, M. Cheng, Q. Yan, and X. Tang, "Placement delivery array design for coded caching scheme in D2D networks," *IEEE Trans. Commun.*, vol. 67, no. 5, pp. 3388–3395, May 2019.
[42] J. Zhu, Q. Yan, C. Qi, and X. Tang, "A new capacity-achieving private information retrieval scheme with (almost) optimal file length for coded servers," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1248–1260, 2020.
[43] G. Aupy, A. Benoit, B. Goglin, L. Pottier, and Y. Robert, "Co-scheduling HPC workloads on cache-partitioned CMP platforms," *Int. J. High Perform. Comput. Appl.*, vol. 33, no. 6, pp. 1221–1239, Nov. 2019.
[44] J. Shi, N. Ueter, G. von der Bruggen, and J.-J. Chen, "Partitioned scheduling for dependency graphs in multiprocessor real-time systems," in *Proc. IEEE 25th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2019, pp. 1–12.
[45] M. Yang, W.-H. Huang, and J.-J. Chen, "Resource-oriented partitioning for multiprocessor systems with shared resources," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 882–898, Jun. 2019.
[46] A. Bertout, J. Goossens, E. Grolleau, and X. Poczekajlo, "Template schedule construction for global real-time scheduling on unrelated multiprocessor platforms," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 216–221.
[47] S. Asta, E. Özcan, and A. J. Parkes, "Batched mode hyper-heuristics," in *Learning and Intelligent Optimization*, G. Nicosia and P. Pardalos, Eds. Berlin, Germany: Springer, 2013, pp. 404–409.
[48] E. Lara-Cárdenas, A. Silva-Gálvez, J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, and H. Terashima-Marín, "Exploring reward-based hyper-heuristics for the job-shop scheduling problem," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2020, pp. 3133–3140.
[49] Y. Pylyavskyy, A. Kheiri, and L. Ahmed, "A reinforcement learning hyper-heuristic for the optimisation of flight connections," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2020, pp. 1–8.
[50] A. Muklason, G. B. Syahrani, and A. Marom, "Great deluge based hyper-heuristics for solving real-world university examination timetabling problem: New data set and approach," *Procedia Comput. Sci.*, vol. 161, pp. 647–655, Jan. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919318794
[51] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A reinforcement learning–great-deluge hyper-heuristic for examination timetabling," *Int. J. Appl. Metaheuristic Comput.*, vol. 1, no. 1, pp. 39–59, Jan. 2010, doi: 10.4018/jamc.2010102603.
[52] F. Garza-Santisteban, I. Amaya, J. Cruz-Duarte, J. C. Ortiz-Bayliss, E. Ozcan, and H. Terashima-Marin, "Exploring problem state transformations to enhance hyper-heuristics for the job-shop scheduling problem," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2020, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/9185709/
[53] X. F. C. Sanchez-Diaz, J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, S. E. Conant-Pablos, and H. Terashima-Marin, "A preliminary study on feature-independent hyper-heuristics for the 0/1 knapsack problem," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2020, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/9185671/
[54] A. Kheiri and E. Keedwell, "A sequence-based selection hyper-heuristic utilising a hidden Markov model," in *Proc. Annu. Conf. Genetic Evol. Comput. (GECCO)*, Jul. 2015, pp. 417–424.
[55] A. Kheiri, E. Özcan, R. Lewis, and J. Thompson, "A Sequence-based selection hyper-heuristic: A case study in nurse rostering," in *Proc. 11th Int. Conf. Pract. Theory Automated Timetabling (PATAT)*, 2016, pp. 503–505.
[56] A. Kheiri, "Heuristic sequence selection for inventory routing problem," *Transp. Sci.*, vol. 54, no. 2, pp. 302–312, Mar. 2020.
[57] L. Ahmed, C. Mumford, and A. Kheiri, "Solving urban transit route design problem using selection hyper-heuristics," *Eur. J. Oper. Res.*, vol. 274, no. 2, pp. 545–559, 2019, doi: 10.1016/j.ejor.2018.10.022.
[58] W. B. Yates and E. C. Keedwell, "An analysis of heuristic subsequences for offline hyper-heuristic learning," *J. Heuristics*, vol. 25, no. 3, pp. 399–430, Jun. 2019, doi: 10.1007/s10732-018-09404-7.
[59] A. Gaier, A. Asteroth, and J.-B. Mouret, "Data-efficient design exploration through surrogate-assisted illumination," *Evol. Comput.*, vol. 26, no. 3, pp. 381–410, Sep. 2018. [Online]. Available: https://www.mitpressjournals.org/doi/abs/10.1162/evco_a_00231
[60] E. Samuelsen and K. Glette, "Multi-objective analysis of MAP-elites performance," 2018, *arXiv:1803.05174*. [Online]. Available: http://arxiv.org/abs/1803.05174
[61] E. Dolson, A. Lalejini, and C. Ofria, "Exploring genetic programming systems with MAP-Elites," in *Genetic Programming Theory and Practice XVI*. Cham, Switzerland: Springer, 2019, pp. 1–16.
[62] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, and H. Terashima-Marín, "A primary study on hyper-heuristics to customise metaheuristics for continuous optimisation," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2020, pp. 1–8.
[63] G. L. Pappa, G. Ochoa, M. R. Hyde, A. A. Freitas, J. Woodward, and J. Swan, "Contrasting meta-learning and hyper-heuristic research: The role of evolutionary algorithms," *Genetic Program. Evolvable Mach.*, vol. 15, no. 1, pp. 3–35, Mar. 2014.
[64] J. Lever, M. Krzywinski, and N. Altman, "Points of significance: Model selection and overfitting," *Nature Methods*, vol. 13, no. 9, pp. 703–704, 2016.
[65] J. R. Woodward, C. G. Johnson, and A. E. I. Brownlee, "Connecting automatic parameter tuning, genetic programming as a hyper-heuristic, and genetic improvement programming," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2016, pp. 1357–1358.
[66] I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. C. Coello, "Tailoring instances of the 1D bin packing problem for assessing strengths and weaknesses of its solvers," in *Parallel Problem Solving From Nature (PPSN XV)* (Lecture Notes in Computer Science), vol. 11101, A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, Eds. Cham, Switzerland: Springer, 2018, pp. 373–384, doi: 10.1007/978-3-319-99259-4_30.

[67] L. F. Plata-González, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. C. Coello, "Evolutionary-based tailoring of synthetic instances for the knapsack problem," *Soft Comput.*, vol. 23, no. 23, pp. 12711–12728, Dec. 2019, doi: 10.1007/s00500-019-03822-w.

[68] I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, and H. Terashima-Marin, "Hyper-heuristics reversed: Learning to combine solvers by evolving instances," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 1790–1797. [Online]. Available: https://ieeexplore.ieee.org/document/8789928/

[69] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic, and E. K. Burke, "HyFlex: A benchmark framework for cross-domain heuristic search," in *Evolutionary Computation in Combinatorial Optimization*, J.-K. Hao and M. Middendorf, Eds. Berlin, Germany: Springer, 2012, pp. 136–147.

MELISSA SÁNCHEZ was born in Culiacan, Sinaloa, Mexico, in 1996. She received the B.Sc. degree in mechatronics engineering from the Tecnologico de Monterrey, in 2020.

From 2018 to 2020, she was a part of the Research Group with Strategic Focus in Autonomous Vehicles, Tecnologico de Monterrey. From 2019 to 2020, she was a Research Assistant with the Research Group with Strategic Focus in Intelligent Systems, Tecnologico de Monterrey. She also has experience in the industry as she worked at John Deere as a Product Design Engineer. Since 2020, she has been working as a CAD Software Engineer at Kalypso: A Rockwell Automation Company. Her research interests include underwater autonomous vehicles (UAVs), combinatorial optimization problems solved through hyper-heuristics, and intelligent systems.

JORGE M. CRUZ-DUARTE (Member, IEEE) was born in Ocaña, N.S., Colombia, in 1990. He received the B.Sc. and M.Sc. degrees in electronic engineering from the Universidad Industrial de Santander, Bucaramanga, Santander, Colombia, in 2012 and 2015, respectively, and the Ph.D. degree in electrical engineering from the Universidad de Guanajuato, Mexico, in 2018. He was a Postdoctoral Fellow with the Research Group With Strategic Focus in Intelligent Systems, Tecnológico de Monterrey, Mexico, from 2019 to 2021, where he is currently a Research Professor with the School of Engineering and Sciences.

His research interests include data science, optimization, mathematical methods, thermodynamics, digital signal processing, electronic thermal management, and fractional calculus.

JOSÉ C. ORTIZ-BAYLISS (Member, IEEE) was born in Culiacan, Sinaloa, Mexico, in 1981. He received the B.Sc. degree in computer engineering from the Universidad Tecnologica de la Mixteca, in 2005, the M.Sc. degree in computer sciences and the Ph.D. degree from the Tecnologico de Monterrey, in 2008 and 2011, respectively, the M.Ed. degree from the Universidad del Valle de Mexico, in 2017, the B.Sc. degree in project management from the Universidad Virtual del Estado de Guanajuato, in 2019, and the M.Ed.A. degree from the Instituto de Estudios Universitarios, in 2019.

He is currently an Assistant Research Professor with the School of Engineering and Sciences, Tecnologico de Monterrey. His research interests include computational intelligence, machine learning, heuristics, metaheuristics, and hyper-heuristics for solving combinatorial optimization problems. He is a member of the Mexican National System of Researchers, the Mexican Academy of Computing, and the Association for Computing Machinery.

IVAN AMAYA (Member, IEEE) was born in Bucaramanga, Santander, Colombia, in 1986. He received the B.Sc. degree in mechatronics engineering from the Universidad Autónoma de Bucaramanga, in 2008, and the Ph.D. degree in engineering from the Universidad Industrial de Santander, in 2015.

From 2016 to 2018, he was a Postdoctoral Fellow with the Research Group with Strategic Focus in Intelligent Systems, Tecnologico de Monterrey. Since 2018, he has been a Research Professor with the School of Engineering and Sciences, Tecnologico de Monterrey. His research interests include numerical optimization of both continuous and discrete problems, through the application of heuristics, metaheuristics, and hyper-heuristics. For the latter, he focuses on finding new ways of using feature transformations for improving performance. He is a member of the Mexican National System of Researchers, the Mexican Academy of Computing, and the Association for Computing Machinery.