Transferability in Optimization Via Multidomain Hyperheuristics

Etor Arza, Ekhiñe Irurozki, Josu Ceberio, Aritz Pérez

***Abstract*—The field of hyperheuristics studies the automatized generation and parametrization of optimization algorithms. According to the literature, one of the limitations of hyperheuristics is that they are often specific to an optimization problem.**
In this paper, we propose a general multi-domain hyperheuristic framework applicable to a wide range of optimization problems with different search spaces. The hyperheuristic has a neural network controller that learns to solve a given optimization problem. The controller can then transfer the behavior learned during training to solve a previously unseen problem.
Moreover, we propose two problem analysis methods that embed a set of optimization problems into the real space. These two embeddings are generated via the performance and the behavior of the hyperheuristic and are independent of the problem domain. An experimental section in four problem sets reveals that the proposed embeddings are correlated with the properties of the optimization problems.

***Index Terms*—Optimization, Hyperheuristics, Transfer Learning**

I. Introduction

Hyperheuristic algorithms automatize the generation and selection of heuristic and metaheuristic algorithms [1]. Classical proposals approached this task by focusing on a certain optimization problem [2]–[7]. However, one of the limitations of classical hyperheuristics is that, once designed, they are only directly applicable to the problem they were originally designed for.

Overcoming this limitation, great progress has been made in "multidomain hyperheuristics", where the same hyperheuristic can be applied to different optimization problems [8], [9]. In this context, HyFlex [10] introduced a popular benchmark platform, where six types of combinatorial optimization problems were proposed, together with a set of low-level heuristic algorithms. Nonetheless, the proposal is only applicable to combinatorial type problems [10].

Recently, Neural Combinatorial Optimization [11], [12] (NCO) has been a major breakthrough in the heuristic optimization of combinatorial problems. It studies the use of neural networks to read the problem data and based on this data, optimize a combinatorial optimization problem [11], [13], [14]. Initial approaches relied on the same idea of constructive heuristics, iteratively building a solution by adding one item at a time [15], or returning a full solution in one shot [16], [17]. However, more recent models propose iteratively improving a solution by applying local search operators (inferred by the model), in a specific position of the solution (also chosen by the neural network) [18]–[20]. These later NCO approaches fit the description of hyperheuristics and, thus, can be considered so.

Unfortunately, NCO approaches are specific to the type of problem being solved: they are designed to solve combinatorial problems with specific problem structures. Consequently, they cannot always be directly applied to other problems, even when the search space of the problems is the same. Moreover, there are many problems with unavailable problem data (for example black box problems [21] or problems with the objective function involving simulations). This makes the applicability of NCO approaches limited.

Revising the literature in the continuous domain, some hyperheuristics have been proposed [22], [23], although to a lesser extent [24]. Some approaches in this domain have focused on algorithm selection [25], improving the parameter control of existing heuristics [26], or the discovery of new variants of existing heuristics [27]. But, as far as we are aware, no framework has been able to generalize enough to close the gap between combinatorial and continuous optimization problems.

Contribution

In this paper, extending the preliminary work by Arza et al. [28], we propose a neural network-based multidomain hyperheuristic framework that overcomes the limitations discussed above. The contribution of this paper is a multidomain hyperheuristic framework applicable to different optimization problems with *different search spaces*, both in the combinatorial and the continuous domains. The proposed approach learns to control and combine existing heuristics to solve optimization problems. It does not take into account the parametrization of the objective function, and instead, the hyperheuristic learns to solve optimization problems through the evaluation of the objective function (useful for black box problems).

As a consequence of the adaptation capability of the hyperheuristic, its behavior and performance vary depending on the optimization problem in which it is applied. By analyzing these variations, we propose two methods to embed a set of optimization problems into the real space. Even though the embeddings are generated via the performance and behavior of the hyperheuristic, we show (Section IV-C) that they are correlated with the properties of the optimization problems themselves. What is more, this analysis is domain agnostic:

Etor Arza, Ekhiñe Irurozki, and Aritz Pérez are with the Basque Center for Applied Mathematics (BCAM)

Josu Ceberio is with the Department of Computer Science and Artificial Intelligence at the University of the Basque Country (UPV/EHU)

Manuscript submitted December 7, 2022.

the methodology is exactly the same regardless of the problem domain.

Organization

The rest of the paper is organized as follows: the next section introduces the multidomain hyperheuristic framework. Afterward, in Section III the proposed optimization problem embedding methodology is described. Then, in Section IV, we carry out an experimental study. Finally, Section V concludes the paper and gives research lines for future work.

II. Multidomain hyperheuristic

The proposed hyperheuristic framework is a population-based algorithm with a neural network-based model, called controller, that decides how the solutions are modified at each iteration. The controller is a trainable model with three interconnected parts, that together, modify solutions according to a previously learned behavior. To achieve domain generality, the controller (see Figure 1 for a broader look) has two problem-specific parts that depend on the domain of the problem being solved: an encoder and a decoder. These need to be specified before applying the hyperheuristic and their purpose is to bridge the search space of the optimization problem with the flexible part of the controller: the trainable neural network. In the following, we introduce the dynamics of the proposed hyperheuristic framework, also depicted in Figure 1.

The hyperheuristic starts with a set of solutions generated uniformly at random. Then, for each solution σ_i, the encoder gathers information about the optimization state (for example how similar is σ_i to the rest of the solutions, or the computation budget left). This information is saved into a real valued vector $X \in \mathbb{R}^p$ that we named *feature vector*. A trainable neural network φ then maps X into $Y \in \mathbb{R}^q$, generating a real value vector that we named *response vector*. Then, based on the response vector, the decoder (a meta-operator parameterized[1] by a real valued vector) modifies each solution in the population. This process is repeated for each solution σ_i in the population. Then, if the stopping criteria are met, the process is terminated and the best found solution is reported. Otherwise, the process is repeated with the new population of solutions replacing the old ones.

This hyperheuristic framework is general and can be applied to different types of optimization problems with different search spaces. In this sense, the decoder and the encoder bridge the search space with the space in which the neural network operates (the neural network φ is mapping with n real inputs and m real outputs). The decoder/encoder pair is the only problem-specific component of the hyperheuristic, and it needs to be compatible with the search space of the optimization problem. If an encoder/decoder pair can be defined for a problem, then the hyperheuristic is applicable to that problem.

In order to apply the hyperheuristic for the optimization of a problem, it needs to be trained first. In this sense, we identify two different stages A) the training stage and B) the application stage. In what follows, we carefully describe each of these stages.

[1]For example, a decoder could be a mutation operator with a variable mutation rate, adjusted via a response vector of length one. Or it could be simply choosing an operator (from a set of operators) with $\text{argmax}\{Y\}$.

Algorithm 1: Training the neural network with NEAT [29]

Input:
I: The training optimization problem.
t_{train}: The maximum training time.

1 $\bar{\varphi} = (\varphi_1, ..., \varphi_m) \leftarrow$ randomly initialize a set of neural networks
2 **while** $t < t_{train}$ **do**
3 **for** $j = 1, ..., m$ **do**
4 $g_j \leftarrow$ optimize problem I with the hyperheuristic, with the neural network φ_j in the controller
5 $\bar{\varphi} \leftarrow$ select, crossover and mutate the neural networks $\bar{\varphi}$ based on the computed scores g_j
6 t $\leftarrow$ elapsed_time
7 res $\leftarrow \varphi_j \in \bar{\varphi}$ with the highest g_j
8 **return** res

A. Training stage

The training of the hyperheuristic involves training the neural network in the controller. The neural network is trained in a reinforcement learning setting with the Neuroevolution of Augmenting Topologies (NEAT) [29] algorithm. Specifically, given an optimization problem, NEAT tries to find the neural network design that maximizes a score obtained from the application of the hyperheuristic in the problem, with the neural network in the controller (see Appendix B for additional details).

The training stage (Algorithm 1) starts with an initial random[2] set of m neural networks $\bar{\varphi} = \{\varphi_1, ..., \varphi_m\}$ (line 1), and the score of the neural networks is computed as the best objective function value they obtain when being used to optimize the given optimization problem I (line 4)[3]. Next, the most promising neural networks are chosen and combined, creating new neural networks (line 5). Finally, mutations are applied to the new neural networks and their performance is once again tested, concluding the current generation. This process is repeated until a stopping criterion is reached (line 2). The training stage concludes when the best neural network in the last generation is returned (lines 7-8).

B. Application stage

Once the controller is trained, the trained hyperheuristic can be applied to optimize other unseen instances of the same problem. In addition, it can also be applied to different problems defined in the same domain.

[2]A random neural network refers to a single hidden layer neural network with the weights initialized uniformly at random in the interval [-1,1].

[3]It is worth mentioning that, given an optimization problem, computing the objective function value of a neural network implies optimizing the given instance with the hyperheuristic algorithm using this neural network in the controller. Due to the random nature of the algorithm, several executions are averaged.

Encoder for real problems

- If a new best solution was recently found or not.
- Distance to best solution.
- Distance to closest solution.
- Distance to the average of the solutions in the population.
- Relative ranking of the objective value of solution σ_i with respect to the rest of the solutions.
- Number of evaluations left.

Encoder for permutation problems

- If σ_i is a local optima or not.
- The level of similarity among solutions in the population.
- Relative ranking of the objective value of solution σ_i with respect to the rest of the solutions.
- Number of evaluations left.

Diagram I: The information gathered by the encoder into the feature vector X.

Decoder for real problems

- Move σ_i towards the best solution.
- Move σ_i towards the best solution that σ_i has visited.
- Move σ_i towards the direction of the last movement.

Decoder for permutation problems

- Moving σ_i towards or away from a close solution.
- Moving σ_i towards or away from the best solution.
- Randomly initialize σ_i.
- Apply local search to σ_i.
- Accept a worse solution only with a certain probability.

Diagram II: The modifications applied by the decoders, given the feature vector Y.

C. Cases of study

To validate the proposed approach, in this paper we considered four problem sets in two different search spaces: the space of permutations of size n and $\mathbb{R}^n$. Therefore, we proposed two encoder/decoder pairs, one for each search space. In Diagram I, we summarize the information that the encoder writes into the feature vector for the two implementations considered in this paper. The interested reader can refer to Appendix C-A for further details on the encoders. Similarly, in Diagram II, we summarize how the decoder modifies the solution σ_i for the two implementations considered. Further details are available in Appendix C-B.

III. The hyperheuristic framework as a tool to analyze optimization problems

As explained in the introduction, the contribution of this paper goes beyond proposing a domain agnostic hyperheuristic framework. We also show that it is a relevant tool to analyze optimization problems, and embed them into the real space according to their properties. In this section, we introduce two embeddings for a set of optimization problems: one via the performance of the hyperheuristic and another one via its behavior. Note that these two methods are general and not problem or domain specific.

A. Via the performance of the hyperheuristic

The first method involves measuring the performance of the hyperheuristic in the problem set, and how it varies when applied to different problems in the set. In this context, we define the *transferability* as the relative loss of performance of the hyperheuristic applied to a problem (the *test* problem) when it has been trained in another problem. By computing the transferability between every pair of problems in a problem set, we can embed these problems into a $k \times k$ real matrix, which we named *transferability heatmap*.

1) Transferability heatmap: Given a set of optimization problems $A_1, ..., A_k$, the transferability from problem A_i to problem A_j, denoted as $T_{i,j}$, is a real value in the interval $[0, 1]$ that represents the relative performance of the hyperheuristic in the test problem A_j, when it has been trained in problem A_i (a lower transferability indicates better performance). This performance is relative to the performance observed when the hyperheuristic has been trained in another different problem $A_{i'}$, and tested in the same problem A_j. More precisely, given a test problem A_j, we train the hyperheuristic in all the problems $A_1, ..., A_k$, and set $T_{i,j} = 0$ for the training problem A_i that produced the best performance in the test problem A_j, and $T_{i,j} = 1$ for the training problem A_i that produced the worst performance (the rest of the problems get values in between, according to the ranking in performance). We repeat this process for each test problem A_j.

Once we have repeated this process for each test problem A_j, we have measured the transferability $T_{i,j}$ between every possible pair of problems. Next, we sort the indices of the problems with heatmap clustering [30] such that the adjacent columns and rows are as similar as possible from each other (details available in Appendix D). With this, we have generated the transferability heatmap: an embedding of the set of k optimization problems $A_1, ..., A_k$ into a real $k \times k$ matrix.

All the technical details have been omitted, but are available in Appendix D for the interested reader. In addition, the source code to generate the transferability is available in our GitHub repository.

2) Interpretation: Figure 2 shows an example transferability heatmap. If the transferability $T_{i,j}$ is low, the interpretation is "the performance of the hyperheuristic on problem A_j was relatively high when trained on problem A_i". Now we can go a step further by comparing several $T_{i,j}$ to each other.

Let us assume that problems A_1 and A_2 are the same optimization problem. Then, it is likely that $T_{1,j} \approx T_{2,j}$ and $T_{i,1} \approx T_{i,2}$. Consequently, A_1 and A_2 will be placed adjacent

to each other in the transferability heatmap.[4] This suggests that problems that are similar to each other from the point of view of the performance of the hyperheuristic will be placed next to each other in the heatmap.

3) Comparison to other works: Hong et al. [31] proposed measuring the transferability as the average objective value of the hyperheuristic when training and testing two *problem classes*. Our experimental setup on transferability improves the methodology of Hong et al. [31] in three key aspects. Firstly, by defining the transferability with ranks instead of objective values, we can compare across different problems (the magnitude of the transferability is the same for every problem). Secondly, by repeating several measurements of the transferability and computing the average ranks, we can distinguish between noise and the actual difference in performance (this is not possible with average objective values). Thirdly, we take existing optimization problems in the literature and analyze the transferability of the hyperheuristic among them. This allows us to generate a visualization of optimization problems that is correlated with the properties of the problems. See Appendix D-C1 for additional details on related work.

B. Via the behavior of the hyperheuristic

The second analysis method takes into account how the hyperheuristic carries out the optimization process. In this sense, the behavior of the trained hyperheuristic (how the decoder modifies each of the solutions) is determined by the outputs of the neural network (the response vectors). In fact, the set of all the response vectors generated when applying the hyperheuristic in an optimization problem allows us to replicate the optimization process carried out. Therefore, we summarize the behavior of the hyperheuristic by averaging all the response vectors generated when applying the hyperheuristic in an optimization problem.

Given a set of optimization problems $A_1, ..., A_k$, we want to identify the similarities/differences of the problems by looking at the behavior of the hyperheuristic when trained and tested in these problems. To do so, first, we train the hyperheuristic in problem A_i and compute the average response $R(A_i \rightarrow A_j)$ generated when applying the hyperheuristic in problem A_j. We repeat this process for every possible pair of optimization problems $A_i, A_j \in \{A_1, ..., A_k\}$, and we obtain 10 samples for each possible pair.

Next, we fit a Linear Discriminant Analysis [32], [33] (LDA). In Appendix E, we justify why fitting an LDA is suitable for this purpose. The class to fit the LDA is the problem A_i used to train the hyperheuristic. Once the LDA is fitted, we project the average response in each train problem[5] into a 2D space. This generates an embedding of the optimization problems in $\mathbb{R}^2$ (one point for each training problem A_i). We denote this embedding as *the LDA of a problem set*.

Once the embedding has been computed, we can interpret it. The interpretation is simple: the behavior of the hyperheuristic is similar in two problems if they are close to each other in the projected space.

[4] As long as the rest of the $T_{s,j}$ $\forall s \neq 1, 2$ are different to $T_{1,j}$ and $T_{2,j}$ and the equivalent for columns. This will not be true, for example, when every $A_1, ..., A_k$ are the same problem, or the training procedure is unable to adapt the hyperheuristic to the optimization problems.

[5] The average response in each train problem A_i can be computed as $\frac{\sum_{A \in \{A_1, ..., A_k\}} R(A_i \rightarrow A)}{k}$

IV. Experimental study

The purpose of this experimental section is to validate the utility of the proposed hyperheuristic framework. The first part of the experimentation is devoted to showing that the hyperheuristic framework can improve existing heuristics beyond a parameter search approach. Then, the second part focuses on the application of the two optimization problem analysis methods described in the previous section. We apply the two analysis methods and then we take a look at how these analyses correlate with the properties of the optimization problems. The code to reproduce the experiments is available in our GitHub repository at https://github.com/EtorArza/TransfHH.

A. Experimental setting

The experimentation was carried out in four problem sets. In the first problem set i), we considered twelve classical continuous optimization problems chosen from the virtual library of simulation experiments [34]. Furthermore, in the second problem set ii), we used the continuous problem instance generator from Rökkönen et al. [35] to experiment on optimization problems with a different number of local optima. Regarding the combinatorial domain, in the third problem set iii) four permutation problems were considered: the Traveling Salesman Problem [36], the Quadratic Assignment Problem [37], the Linear Ordering Problem [38] and the Permutation Flowshop Scheduling Problem [39] and four instances of different sizes and benchmarks were chosen for each problem (in total 16 instances). In the final problem set iv), we considered different instance benchmarks from the Quadratic Assignment Problem. Further details on the four i) - iv) problem sets are available in Appendix A.

The experimentation parameters are listed below. Note that these parameters are the same regardless of the problem set considered in each case.

Hyperheuristic parameters

- Population size: 8
- Stopping criterion: 400 evaluations

Training parameters (NEAT)

- Population size: 10^3 (default value [40])
- Stopping criteria: 4 days or 2000 generations (stop when either criterion is met)

Testing parameters

- Executions averaged: 10^4

B. Comparison to classical parameter search

The goal of this experiment is to measure the capability of the hyperheuristic of improving an existing heuristic when compared to parameter tuning. To do so, we compare the hyperheuristic with the standard particle swarm optimization (PSO) algorithm [41] (see Appendix C-B1 for details) doing a grid search on its parameters. This choice is not arbitrary, as the decoder considered in the continuous case applies the PSO algorithm, with the parameters adjusted with the output of the neural network. We limit the comparison to problem sets i) and ii) because in the discrete case, the hyperheuristic with the decoder for permutations considered in this paper is not a parametrization of an existing heuristic, and thus, a direct comparison is not possible.

Before measuring the performance of the PSO algorithm, its three parameters were adjusted in the interval $[-1.96, 1.96]$ with a grid search approach. The parameter space was discretized in 20 slices for each dimension, and the best combination of parameters was chosen for each problem. Similarly, the hyperheuristic was trained in each of the problems before testing its performance.

The average objective values are shown in Table I. We can see that the hyperheuristic obtains better (higher) result in all problems. Applying the sign hypothesis test, with a chosen $\alpha = 0.05$, we reject the null hypothesis and accept that the hyperheuristic performs better than the heuristic in both problem sets ($p = 0.00049$ and $p = 0.0156$ respectively). These results point out that the proposed hyperheuristic was able to improve the PSO algorithm beyond a parameter tuning approach.

We hypothesize that the hyperheuristic has a better performance than the heuristic because the search space of the hyperheuristic is a superset of the search space of the parameter search algorithm. In other words, if we limit the hyperheuristic to having the same response during the whole optimization process, then it is the same algorithm as the heuristic algorithm. Unlike the parameter search approach, the hyperheuristic can deal with a wider range of behaviors that take into account the state of the optimization.

problem set i)

Problem index	Heuristic	Hyperheuristic
A_1	-62.7274	**-2.8240**
A_2	-10277.2705	**-528.3484**
A_3	41.9383	**119.1122**
A_4	-4.9798	**-0.4369**
A_5	-1.5273	**-1.0013**
A_6	-2072.1211	**-76.5153**
A_7	9.2708-08	**0.0182**
A_8	2753.5490	**4479.4734**
A_9	-244.9184	**-65.9289**
A_{10}	-86.2573	**-9.1784**
A_{11}	-216.3577	**-11.8173**
A_{12}	3.1433	**12.5266**

problem set ii)

Number of local optima	Heuristic	Hyperheuristic
1	-77.1428	**-0.4914**
2	-44.9687	**-0.8423**
4	-20.9279	**-0.6745**
8	-11.3950	**-0.6639**
16	-11.2409	**-1.0411**
32	-12.4135	**-4.1041**
64	-8.2742	**-4.6253**

Table I: Comparison of the heuristic and the hyperheuristic in the problem sets i) and ii). Higher is better (highlighted in bold).

C. Analyzing optimization problems

The goal of this part of the experimentation is to validate the two optimization problem analysis methods introduced in Section III. In addition, we take a close look at the correlation that these methods have with respect to the properties of the optimization problems in each of the problem sets. It is important to note that the properties & types of the optimization problems are not taken into account when applying these two methods. Consequently, the observed results arise from the differences in the performance and behavior of the hyperheuristic when applied to these problems.

We start with the analysis of the transferability heatmap generated via the performance of the hyperheuristic.

1) Via the performance of the hyperheuristic: In this first part, we apply the methodology proposed in Section III-A and analyze the four problem sets via the performance of the hyperheuristic.

Problem set i)

The transferability heatmap for this problem set is shown in Figure 3. First, observe that the values in the diagonal are very low (lower means better performance). This is the expected result: the hyperheuristic will obtain the best possible objective value when it is trained and tested in the same problem.

Recall that, when generating the heatmap, columns and rows are sorted together via hierarchical biclustering [30], such that adjacent columns and rows are as similar as possible to each other (see Section III-A for a detailed explanation). Taking into account the similarities among adjacent columns and rows in the heatmap, we defined three problem clusters (see Figure 4). We defined the first and largest cluster as the set of optimization problems $\{A_6, A_{11}, A_2, A_1, A_5\}$.

We defined the second cluster as the set $\{A_8, A_7, A_4, A_3\}$. The transferability of A_3 is very different from the rest. The hyperheuristic trained on problem A_3 will perform well on problem A_3, and it will perform poorly in the rest of the problems. However, observe that

$T_{4,3} > T_{i,3}\ \forall i \neq 3, 4$ and $T_{3,4} > T_{3,i}\ \forall i \neq 3, 4$.

From this, we can deduce that problems A_3 and A_4 are related to each other. Interestingly enough, if we look at their contour plots in Figures 3 and 4 respectively, we see that they have a rather similar shape, with both of them having a gradient in

the whole space toward the global optima, located near one of the corners.

We defined the third cluster as the set $\{A_{10}, A_{12}, A_9\}$. A_9 is very different to $\{A_{10}, A_{12}\}$ but it is still related to A_{10} as $T_{9,10} > T_{9,i}\ \forall i \neq 9, 10$. In this sense, A_{10} is the most similar problem to A_9. Therefore, it would be possible to consider $\{A_9\}$ as the fourth cluster, but we placed it together with $\{A_{10}, A_{12}\}$ to avoid a single element cluster.

The problems within each cluster have similar properties. In the first cluster, we find problems with 1) a quadratic component with big importance in the objective function and 2) the global optimum at the center of the search space[6]. The problem A_{12} could also fit in this cluster, as it also has a quadratic component with the optimum near the center of the search space (see Figure 12). Problems in the second cluster have the global optima in a corner of the search space (unlike the rest of the problems). Finally, the problems in the third cluster have many local optima that are "comparably good to each other" in a large area of the search space, and have the global optimum in the center of the search space.

In conclusion, the methodology proposed in Section III-A generated a transferability heatmap for problem set i) that was very correlated with the properties of the optimization problems.

Problem set ii)

The transferability heatmap for the second problem set is shown in Figure 5. Unlike in problem set i), in this problem set there seem to be no differences in transferability: it does not matter the number of local optima that the optimization problems—generated with the by Rökkönen et al. [35] problem generator—have regarding the performance and learning capability of the hyperheuristic. Consequently, the transferability is very similar in all problems.

In conclusion, the proposed methodology identified that there was no adaptation to the number of local optima. We can deduce this from the figure as most of the values are similar and are close to 0.5. It may seem like a trivial result, but measuring the magnitude of the transferability (small in this case) is not trivial. Specifically, we were able to observe this result because of the specific definition of the transferability considered: including the averaging of repeated samples and the use of ranks instead of the objective function directly (explained in detail in Appendix D-C).

Problem set iii)

In Figure 6, we show the transferability heatmap for the third iii) problem set. In this problem set, we have four types of optimization problems (LOP, PFSP, QAP, and TSP) and eight problems instances for each problem type[7]. From now on, "QAP" will be used to refer to any one of the QAPs in problem set iii) (LOPs, PFSPs, and TSPs will be denoted in the same way). In addition, we will use $\neg$ QAP to refer to any one of the problems in problem set iii) that are not QAPs and PROB to refer to any one of the problems in problem set iii).

First, observe that

1) $T_{\text{QAP,QAP}} < T_{\neg\,\text{QAP,QAP}}$ and
2) $T_{\text{QAP},\neg\,\text{QAP}} > T_{\text{QAP},\neg\,\text{QAP}}$.

We deduce 1) from columns 1-8 and 2) from rows 1-8 in Figure 6. The interpretations are 1) the hyperheuristic performs better in QAPs when it is trained in QAPs and 2) the hyperheuristic performs worse in non-QAPs when it is trained in QAPs. The rest of the problems do not satisfy an analogous version of 1) and 2). Therefore, the QAP is the problem that is the most *different* from the rest of the problems, from the point of view of the performance of the hyperheuristic.

Now let us consider the order of the problems in the heatmap. Notice that the QAPs are placed together in the first 8 rows/columns. Next, we have the 8 PFSPs, which would all be together, but a LOP has slipped between them. Notice that the LOPs is right between the problems tai20_20_0.pfsp, tai20_20_1.pfsp, and the rest of the PFSPs. A possible explanation is that these two PFSPs have a lower number of tasks (20) than the rest of the PFSPs (50 and 100) in problem set iii).

Finally, notice that the TSPs and LOPs are mixed. This means that, from the point of view of the performance of the considered hyperheuristic, a TSP and a LOP are similar. Ceberio et al. [42] also reached a similar conclusion in their work with multi-start local search.[8]

In conclusion, the transferability heatmap for problem set iii) successfully grouped the problem types into three groups: QAP, PFSP, and the rest of the problems. The analysis also pointed out that the performance of the hyperheuristic is similar in TSPs and LOPs, which is consistent with previous findings [42].

Problem set iv)

In Figure 7, we show the transferability heatmap for the fourth iv) problem set. In this problem set, we study three QAP instance types: *taixxa*, *taixxb* and *sko* (A_i, B_i and C_i respectively in the figure). As we can see in the figure, there is a big difference between *sko* instances and the rest (rows/columns 15-21). Regarding the order, *sko* instances are all next to each other, while *taixxa* and *taixxb* instances are mixed. In addition, there seems to be almost no difference in transferability between *taixxa*, *taixxb* instances.

The proposed analysis clearly suggests a difference between *sko* instances and the rest, but does not suggest differences among the rest of the instances. This should not be interpreted as *taixxa* and *taixxb* instances being similar: they are only similar from the point of view of the performance of the hyperheuristic.

[6]See the contour plots in Appendix A - i).

[7]See Appendix A - iii) for details on the problems in this problem set.

[8]Ceberio et al. [42] found out that multi-start local search with the insert neighborhood [43] outperforms the other two multi-start algorithms studied in the TSP and LOP, but not in the PFSP or the QAP. Note that the hyperheuristic considered in this paper also uses the insert operator and insert local search (see Appendix C.B.2).

2) Via the behavior of the hyperheuristic: In this second part, we analyze the optimization problems with the LDA of the problem sets, generated via the behavior of the hyperheuristic following the methodology introduced in Section IV-C2. The experimentation is carried out in three out of the four problem sets. We skipped problem set ii) because, as we saw in the previous part of the experimentation, the hyperheuristic performs the same in these optimization problems, hence looking at the behavior makes no sense in this case.

Problem set i)

Figure 8a shows the LDA of problem set i). We colored the data points based on the three problem clusters defined in Section IV-C1. As seen in the Figure, these three clusters are linearly separable. This means that for problem set i) there is a high correlation between the two analysis methods carried out. The results validate the conclusions drawn in the previous part of the experimentation (Section IV-C1) on the same problem set: both analysis methods are correlated with the properties of the optimization problems.

Problem set iii)

Figure 8b shows the LDA for problem set iii). Firstly, notice that all of the QAPs are linearly separable from the rest. Also, most of the PFSPs are clustered together, except for two of them. Finally, the LOPs and TSPs are mixed together, although it seems that LOPs are somehow surrounding the TSPs.

In the transferability heatmap for this problem set, we also observed that the QAP problems were the most different from the rest of the problems and that LOP and TSP problems were similar to each other. Hence, both embeddings identify this similarity between LOP and TSP problems.

Problem set iv)

Figure 8c shows the LDA for problem set iv). The three types of instances are quite visibly separated from each other. It could be argued that *Taixxa* and *Taixxb* are less separated from each other as they are located on the same vertical axis, and *Sko* instances are further away in the horizontal axis.

In contrast, in the transferability heatmap of this problem set, *Sko* instances were very different from the other instances but *Taixxa* and *Taixxb* were indistinguishable from each other from the point of view of the transferability. Interestingly, this suggests that even though performance-wise *Taixxa* and *Taixxb* are similar, the behaviors learned by the hyperheuristic during training are still different.

V. Conclusion and future work

In this paper, we proposed a multidomain hyperheuristic framework applicable to different optimization problems in different domains. The hyperheuristic can be used to improve existing heuristics. Moreover, it can also be used to analyze optimization problems according to their properties.

Specifically, we proposed two embeddings into the real space for a set of problems: one based on the performance of the hyperheuristic and another one based on its behavior. The two methods are domain agnostic: they are not specific to the search space and only require the evaluation of the objective function. In an experimental study on four problem sets, we showed that the two analysis methods are useful to gain new insight on a set of problems. We also showed that the embeddings are correlated with the properties of the optimization problems.

The code to reproduce the experiments and apply the proposed methodology is available in our GitHub repo at https://github.com/EtorArza/TransfHH. In future work, it would be interesting to offer the proposed hyperheuristic framework together with the analysis methods as a python library. This would make it as easy as possible for practitioners to use the proposed framework to improve existing heuristics and analyze optimization problems within the proposed framework.

Acknowledgments

This work is partially supported by the data science and artificial intelligence chair for digitalized industry and services, Telecom Paris, Institut Polytechnique de Paris, and by the Basque Government through the BERC 2022-2025 and the ELKARTEK programs (KK-2020/00049, SIGZE, KK-2021/00065, KONFLOT KK-2022/00100) the Research Groups 2019-2021 (IT1244-19) and by the Spanish Ministry of Economy and Competitiveness, through BCAM Severo Ochoa excellence accreditation SEV-2017-0718 and the research project PID2019-106453GA-I00/AEI/10.13039/501100011033.

References

[1] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, Dec. 2013.

[2] R. E. Keller and R. Poli, "Self-adaptive hyperheuristic and greedy search," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 3801–3808.

[3] J. Grobler, A. P. Engelbrecht, G. Kendall, and V. S. S. Yadavalli, "Alternative hyper-heuristic strategies for multi-method global optimization," in *IEEE Congress on Evolutionary Computation*. Barcelona, Spain: IEEE, Jul. 2010, pp. 1–8.

[4] D. Meignan, A. Koukam, and J.-C. Créput, "Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism," *Journal of Heuristics*, vol. 16, no. 6, pp. 859–879, Dec. 2010.

[5] E. K. Burke, M. R. Hyde, and G. Kendall, "Grammatical Evolution of Local Search Heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 3, pp. 406–417, Jun. 2012.

[6] S. Martin, D. Ouelhadj, P. Beullens, E. Ozcan, A. A. Juan, and E. K. Burke, "A multi-agent based cooperative approach to scheduling and routing," *European Journal of Operational Research*, vol. 254, no. 1, pp. 169–178, Oct. 2016.

[7] V. Stanovov, S. Akhmedova, and E. Semenkin, "Neuroevolution for Parameter Adaptation in Differential Evolution," *Algorithms*, vol. 15, no. 4, p. 122, Apr. 2022.

[8] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Grammatical Evolution Hyper-Heuristic for Combinatorial Optimization Problems," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 6, pp. 840–861, Dec. 2013.

[9] ——, "Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 309–325, 2015.

[10] 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, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 136–147.

[11] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural Combinatorial Optimization with Reinforcement Learning," 2016.

[12] A. I. Garmendia, J. Ceberio, and A. Mendiburu, "Neural Combinatorial Optimization: A New Player in the Field," 2022.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3104–3112.

[14] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer Networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2692–2700.

[15] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac, "Reinforcement learning for solving the vehicle routing problem," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018.

[16] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.

[17] C. K. Joshi, Q. Cappart, L.-M. Rousseau, T. Laurent, and X. Bresson, "Learning TSP requires rethinking generalization," *arXiv preprint arXiv:2006.07054*, 2020.

[18] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[19] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems.." *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[20] A. I. Garmendia, J. Ceberio, and A. Mendiburu, "Neural Improvement Heuristics for Preference Ranking," 2022.

[21] E. Irurozki and M. López-Ibáñez, "Unbalanced mallows models for optimizing expensive black-box permutation problems," in *The Genetic and Evolutionary Computation Conference (GECCO21)*, 2021.

[22] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, and H. Terashima-Marin, "A Primary Study on Hyper-Heuristics to Customise Metaheuristics for Continuous optimisation," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. Glasgow, United Kingdom: IEEE, Jul. 2020, pp. 1–8.

[23] F. Caraffini, F. Neri, and M. Epitropakis, "HyperSPAM: A study on hyper-heuristic coordination strategies in the continuous domain," *Information Sciences*, vol. 477, pp. 186–202, 2019.

[24] N. Pillay and R. Qu, *Hyper-Heuristics: Theory and Applications*, ser. Natural Computing Series. Cham: Springer International Publishing, 2018.

[25] A. Jankovic and C. Doerr, "Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 841–849.

[26] G. Shala, A. Biedenkapp, N. Awad, S. Adriaensen, M. Lindauer, and F. Hutter, "Learning Step-Size Adaptation in CMA-ES," in *Parallel Problem Solving from Nature – PPSN XVI*, T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, and H. Trautmann, Eds. Cham: Springer International Publishing, 2020, vol. 12269, pp. 691–706.

[27] R. Poli, W. B. Langdon, and O. Holland, "Extending particle swarm optimisation via genetic programming," in *European Conference on Genetic Programming*. Springer, 2005, pp. 291–300.

[28] E. Arza, J. Ceberio, A. Pérez, and E. Irurozki, "An adaptive neuroevolution-based hyperheuristic," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 111–112.

[29] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[30] M. Waskom, "Seaborn.clustermap — seaborn 0.11.2 documentation."

[31] L. Hong, J. H. Drake, J. R. Woodward, and E. Özcan, "A hyper-heuristic approach to automated generation of mutation operators for evolutionary programming," *Applied Soft Computing*, vol. 62, pp. 162–175, Jan. 2018.

[32] P. Singh, "Dimensionality reduction approaches," https://towardsdatascience.com/dimensionality-reduction-approaches-8547c4c44334, 2020-08-18, 2020.

[33] C. R. Rao, "The utilization of multiple measurements in problems of biological classification," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 10, no. 2, pp. 159–203, 1948.

[34] S. Surjanovic and D. Bingham, "Virtual library of simulation experiments: Test functions and datasets," Retrieved November 8, 2021, from http://www.sfu.ca/~ssurjano.

[35] J. Rönkkönen, X. Li, V. Kyrki, and J. Lampinen, "A Generator for Multimodal Test Functions with Multiple Global Optima," in *Simulated Evolution and Learning*, X. Li, M. Kirley, M. Zhang, D. Green, V. Ciesielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. C. Tan, J. Branke, and Y. Shi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 5361, pp. 239–248.

[36] D. E. Goldberg and R. Lingle, Jr., "Alleles, Loci and the Traveling Salesman Problem," in *Proceedings of the 1st International Conference on Genetic Algorithms*. L. Erlbaum Associates Inc., 1985, pp. 154–159.

[37] T. C. Koopmans and M. Beckmann, "Assignment Problems and the Location of Economic Activities," *Econometrica*, vol. 25, no. 1, p. 53, Jan. 1957.

[38] T. Schiavinotto and T. Stützle, "The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms," *Journal of Mathematical Modelling and Algorithms*, vol. 3, no. 4, pp. 367–402, 2004.

[39] J. N. Gupta and E. F. Stafford, "Flowshop scheduling research after five decades," *European Journal of Operational Research*, vol. 169, no. 3, pp. 699–711, Mar. 2006.

[40] P. Dougherty, "Accneat," 2014.

[41] A. Engelbrecht and C. Cleghorn, "Recent advances in particle swarm optimization analysis and understanding," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 747–774.

[42] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano, "A review of distances for the Mallows and Generalized Mallows estimation of distribution algorithms," *Computational Optimization and Applications*, vol. 62, no. 2, pp. 545–564, Nov. 2015.

[43] T. Schiavinotto and T. Stützle, "A review of metrics on permutations for search landscape analysis," *Computers & operations research*, vol. 34, no. 10, pp. 3143–3153, 2007.

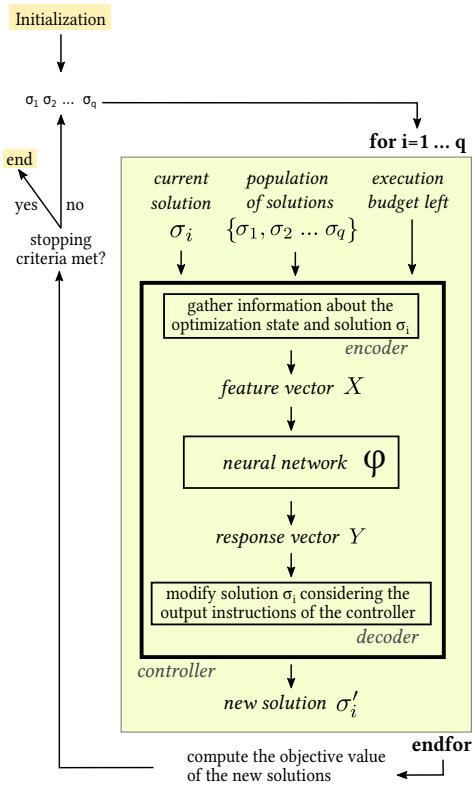


Figure 1: Diagram of the proposed hyperheuristic framework.

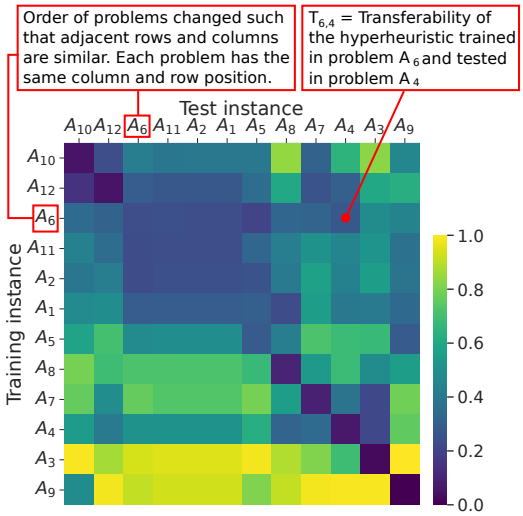


Figure 2: An example transferability heatmap. A darker color implies a lower (better) transferability.

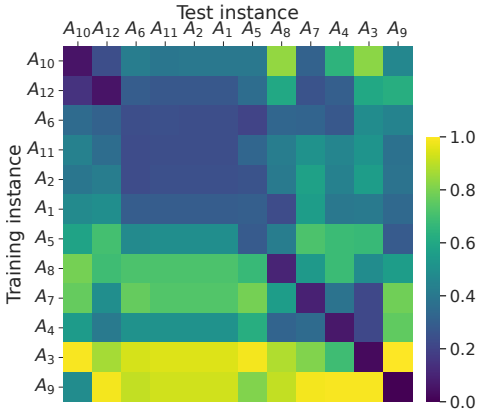


Figure 3: Transferability heatmap of problem set i). Lower (darker) is better.

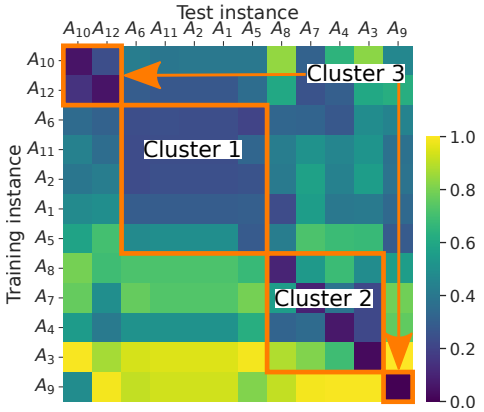


Figure 4: The three proposed problem set clusters for problem set i).

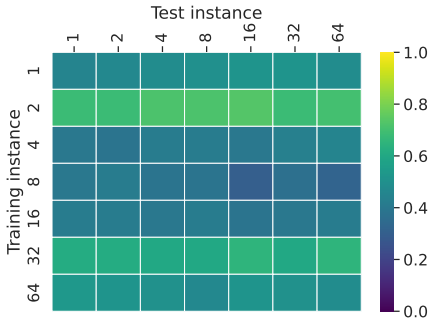


Figure 5: Transferability heatmap of problem set ii). Lower (darker) is better.

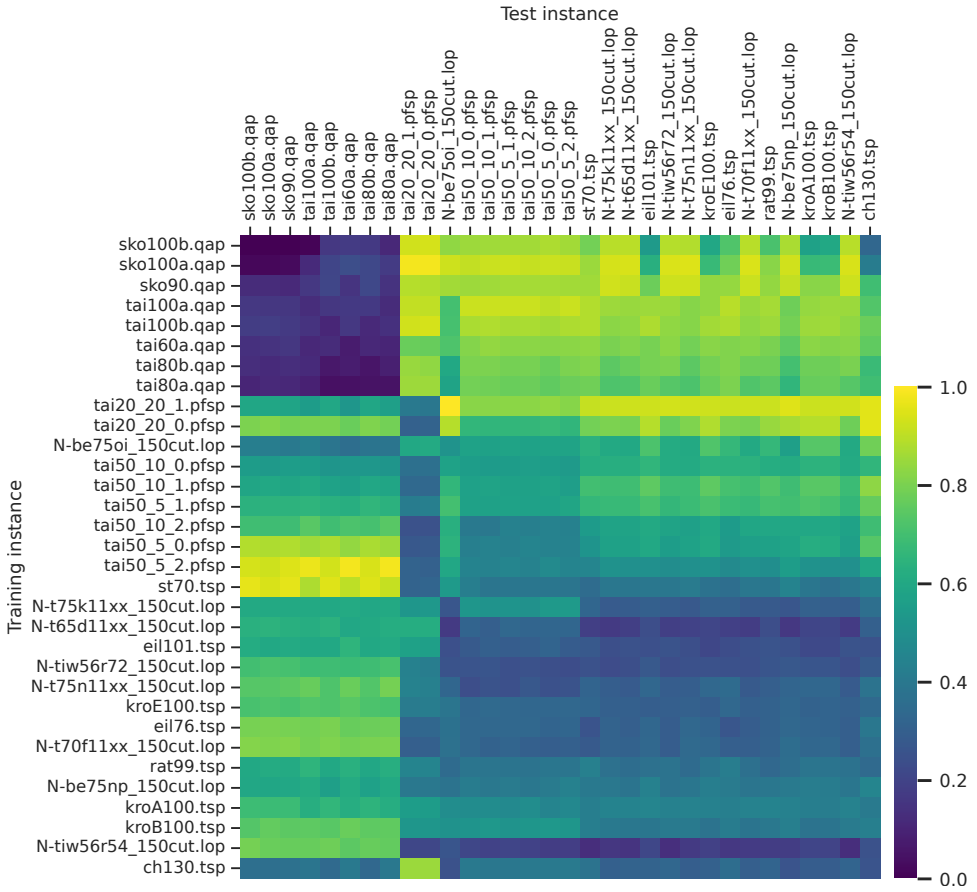


Figure 6: Transferability heatmap of problem set iii). Lower (darker) is better.

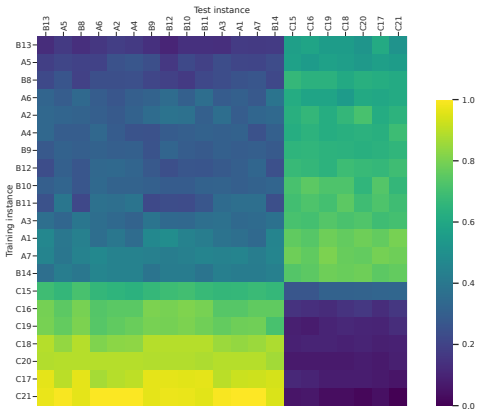
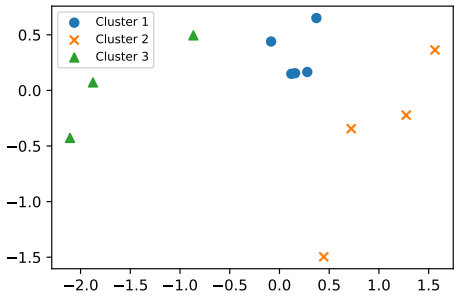
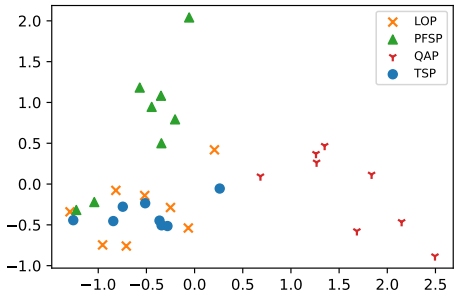


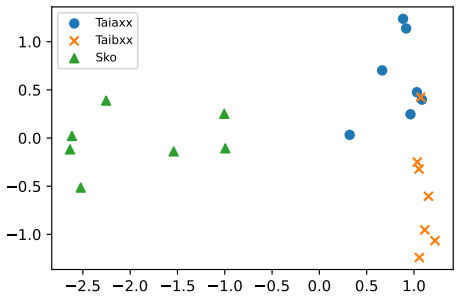
Figure 7: Transferability heatmap of problem set iv). Lower (darker) is better.



(a) problem set i)



(b) problem set iii)



(c) problem set iv)

Figure 8: The LDA for problem sets i), iii) and iv). The colors represent the problem types/clusters within the problem sets. These were added after the LDA was computed. If two optimization problems are close to each other in the LDA, the interpretation is that the hyperheuristic has a similar behavior—*how* it performs optimization—in these two problems.