Tailoring Instances of the 1D Bin Packing Problem for Assessing Strengths and Weaknesses of its Solvers*

Ivan Amaya[1] [0000−0002−8821−7137], José Carlos Ortiz-Bayliss[1] [0000−0003−3408−2166], Santiago Enrique Conant-Pablos[1] [0000−0001−6270−3164], Hugo Terashima-Marín[1] [0000−0002−5320−0773], and Carlos A. Coello Coello[2] [0000−0002−8435−680X]

[1] Tecnologico de Monterrey, School of Engineering and Sciences, Mexico
`{iamaya2, jcobayliss, sconant, terashima}@itesm.mx`
[2] CINVESTAV-IPN (Evolutionary Computation Group), Mexico
`ccoello@cs.cinvestav.mx`

Abstract. Solvers for different combinatorial optimization problems have evolved throughout the years. These can range from simple strategies such as basic heuristics, to advanced models such as metaheuristics and hyper-heuristics. Even so, the set of benchmark instances has remained almost unaltered. Thus, any analysis of solvers has been limited to assessing their performance under those scenarios. Even if this has been fruitful, we deem necessary to provide a tool that allows for a better study of each available solver. Because of that, in this paper we present a tool for assessing the strengths and weaknesses of different solvers, by tailoring a set of instances for each of them. We propose an evolutionary-based model and test our idea on four different basic heuristics for the 1D bin packing problem. This, however, does not limit the scope of our proposal, since it can be used in other domains and for other solvers with few changes. By pursuing an in-depth study of such tailored instances, more relevant knowledge about each solver can be derived.

Keywords: 1D Bin Packing Problem · Genetic Algorithm · Instance generation.

1 Introduction

The Bin Packing Problem (BPP) has been widely studied in the literature [2,3]. Moreover, different kinds of optimization problems can be modelled as BPPs [7]. This has led to a broad array of different solvers for tackling the problem: from

---

* The authors would like to thank CONACyT for the support given through projects No. 241461 and 221551. They would also like to acknowledge the support from the Research Group with Strategic Focus in Intelligent Systems, from Tecnológico de Monterrey.

basic low-level heuristics, going through metaheuristics, and even arriving to high-level solvers known as hyper-heuristics [17]. Even so, instances have remained mostly stable as solvers have evolved throughout the years. Tailoring instances to solvers is thus, needed, so that both of them can evolve in tandem.

There has been an effort for creating instances in different fields. For example, Martello and Toth [11], introduced a set of instances for the binary Knapsack Problem (0/1 KP) with different numbers of items, and a high correlation between weights and profits of the items. Similarly, Zitzler et al. [19] generated data for the multi-objective KP using random integers for weights and profits that has been widely used [18,9,6]. Another example is the OR-Library, which distributes test data for Operations Research (OR) problems [1]. Yet another example corresponds to the Mixed Integer Programming Library (MIPLIB) [7], originally proposed in 1992. The current version dates from 2010 and it covers several types of domains, including Bin Packing. Other works of interest on this regard include [10,14,13], which we do not detail here due to space constraints.

We consider that the aforementioned approach deals with the problem reactively: first creates problems and then verifies the performance of solvers. But, we can change this approach for a proactive one, where we create instances tailored to the attributes of a solver. In this work, we explore the idea of using an evolutionary-based model for creating such an approach. Our objective is finding instances where a solver exhibits a desired behavior, allowing for investigations that revolve around the strengths and weaknesses of such solvers.

This need is not new and it has been explored in recent years. A recurring idea is to use an 'intelligent' generator, based on evolutionary computation, to target a solver. Some authors have achieved quite interesting results. For example, van Hemert [4,5] showed how to use an evolutionary algorithm to detect hard to solve instances in Constraint Satisfaction Problems (CSPs). Smith-Miles et al. [16,15] focused on intentionally creating instances of the Traveling Salesman Problem (TSP) that were easy or hard for certain algorithms.

To the best of our knowledge, no prior works have proposed a tool that allows assessment of the strengths and weaknesses of different solvers, through an evolutionary approach. Based on that, we asked whether it was possible to use a genetic algorithm for producing synthetic 1D bin packing instances under different scenarios. By doing so, we can fill the knowledge gap about instance tailoring for bin packing problems, while at the same time offering a tool for facilitating the study of different solvers under a variety of scenarios. This work contributes by presenting the rationale behind such a tool. Through it, insights about the elements that allow a solver to excel or fail can be more easily obtained.

This manuscript is organized as follows. Section 2 presents an overview of different elements related to the research, focusing on the domain we selected for testing, as well as on the solvers and features of interest. Afterwards, Section 3 briefly presents our proposed tool by describing how it operates. We then move on to summarizing the testing we carried out in Section 4. The corresponding data is given in Section 5. We wrap up our manuscript by laying out our conclusions in Section 6.

2 Fundamentals

This section presents some of the fundamental concepts related to our work. Here, we have restricted our analysis to problems in one dimension, though our model can be expanded to p dimensions without difficulty.

2.1 The 1D Bin Packing Problem

Bin Packing Problems (BPPs) represent the task of packing a set of items into containers with a given capacity (known as bins). The objective is to use as few bins as possible. In the one-dimensional case, objects have a single dimension (e.g., cost, length, time, etc.). Higher dimensional cases are represented by a combination of those, though as aforementioned, we limit ourselves to 1D BPPs. In this work, we consider the dimension of the problem as the length of the item. Hence, the problem can be stated as:

Given a list of objects, and their lengths, as well as a collection of bins with fixed size, find the smallest number of bins that can contain all objects.

One-dimensional BPPs represent NP-hard combinatorial optimization problems, based on the partition problem. Finding the optimal solution is known to be exponentially difficult. Performance of traditional exact methods, such as branch and bound, degrade as the problem grows. Thus, other approaches are required. One of them is to use heuristic methods, representing simple and purpose-specific strategies that can obtain an approximate solution in a short enough time.

2.2 Some Solvers of Interest for the 1D Bin Packing Problem

In this work, we focus on a particular case of online BPPs. We consider problems where information from the whole instance is available, but where packaging is restricted to the first element, e.g., a production line where packaging is carried out by a fixed robot at the end of the line. Hence, all of the following heuristics pack the first item in the list:

- **First Fit Heuristic (BP-FF).** Find all the bins in which the item fits and place it into the lowest numbered one.
- **Best Fit Heuristic (BP-BF).** Find all the bins in which the item fits and place it in the one that leaves the least free space.
- **Worst Fit Heuristic (BP-WF).** Place the item in the bin with the most available space, as long as it fits.
- **Almost Worst Fit Heuristic (BP-AWF).** Similar to BP-WF, but places the item in the second emptiest bin (as long as it can hold it).

We want to stress out that if the item does not fit in any bin, a new one is opened and the item is placed there. Moreover, selecting these heuristics does not limit the scope of our work. In fact, our proposed model can handle different solvers as it only requires knowing how it performs for the instance being tailored.

2.3 Some Features for the 1D Bin Packing Problem

Even though our proposed model is featureless, it can be useful to define a set of features for analyzing the generated instances. We considered some based on the cost (i.e., length) of elements remaining in the instance [8]: Average length (AL), Standard deviation of the length (SL), and Ratio of big pieces (RBP, i.e., the ratio of elements whose length is above half of the bin capacity).

2.4 Some Performance Measures for the 1D Bin Packing Problem

Based on [12], in this work we adopt three metrics for measuring the performance of a solver over a given instance: the number of bins used, the number of completely filled bins, and the average waste per bin.

Consider this brief example, assuming a bin capacity of 10 and that the set of items to be packed is: $3, 8, 7, 4, 9, 1, 6, 2$. Figure 1 shows the solution with each heuristic, and the aforementioned performance metrics. The best solution is given by the best fit heuristic (BP-BF) since it used the least number of bins and filled them completely. Moreover, by only focusing on the number of bins, it would seem that the remaining heuristics are equally good. However, the worst fit heuristic (BP-WF) was unable to completely fill the same number of bins, and thus represents the worst heuristic. In fact, the remaining two heuristics (BP-FF and BP-AWF) yield the same solution, so they are tied.

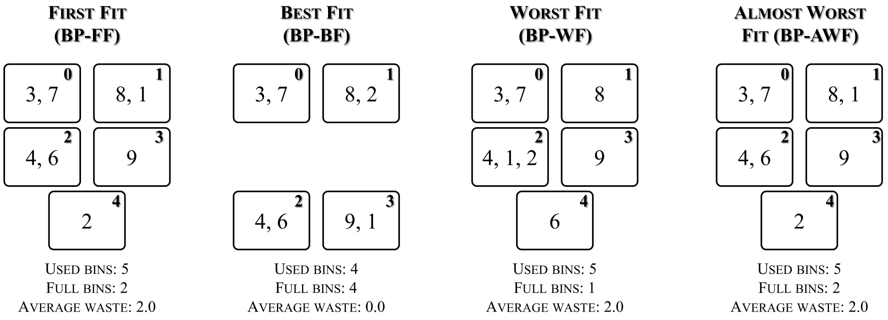


Fig. 1. Solution given by each heuristic, and their performance, for the set of items given by $3, 8, 7, 4, 9, 1, 6, 2$. The number in the upper right corner of each bin represents their ID.

2.5 Instances Used in this Work

Throughout our work, we only consider custom instances. As will be detailed in Section 4, we tailor these instances so that each solver behaves under the conditions defined by the user. The dataset is available upon request.

3 The Proposed Approach

In this work, we propose using a genetic algorithm for directly evolving the parameters of each item in the instance. To do so, a chromosome is represented by a binary string whose size depends on the number of items in the instance and on the maximum value for their length. Figure 2 shows an example of two chromosomes (A and B) with five items each, whose length is given by five bits. Since we are only interested in providing instances with positive length, these chromosomes can represent lengths in the range $[1, 32]$. As can be seen, repetitions are allowed within the instance.

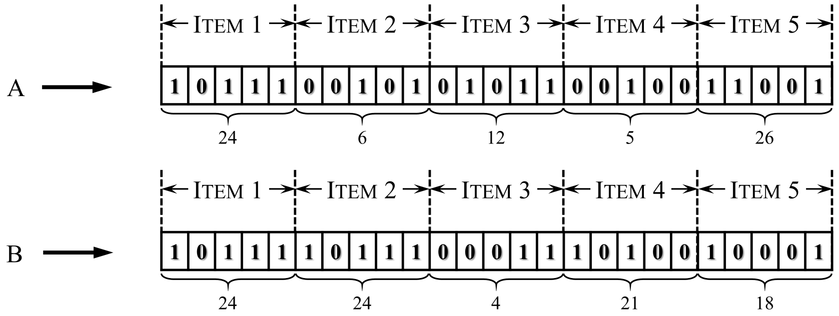


Fig. 2. Two sample chromosomes (A and B) for generating instances with five items and with item lengths in the range $[1, 32]$.

To produce an instance, three arguments are required: the capacity of the bin, C; the number of items for the instance, n; and the maximum length of each item, l_i. The process starts with a population of randomly initialized chromosomes. Afterwards, it keeps evolving guided by an user-defined objective function. This allows instances to be tailored to different means, therefore representing a powerful tool for studying specific traits of different solvers. As will be shown in Section 4, in this work we show different functions that can be used to guide the evolution through different paths.

As evolution progresses, chromosomes will change to reflect item lengths closer to the user requirements. To do so, two new offspring are created at each iteration, using standard genetic operators of selection, crossover, and mutation. The two worst chromosomes are then removed from the population so its size is preserved with each iteration. We want to stress that the number of items, as well as the capacity of the bin, are not encoded within the chromosome. Thus, they remain unchanged throughout the whole process.

4 Methodology

Testing was carried out on an Intel Core i7-6700 processor, with 16 GB of RAM, and running Windows 10 OS. The steady-state GA was tuned through exploratory experiments, omitted for the sake of space: 200 individuals, mutation rate of 0.01, crossover rate of 1.0 and 25000 generations per run (tops). To gather our data, we followed a three-stage methodology as described in Fig. 3.

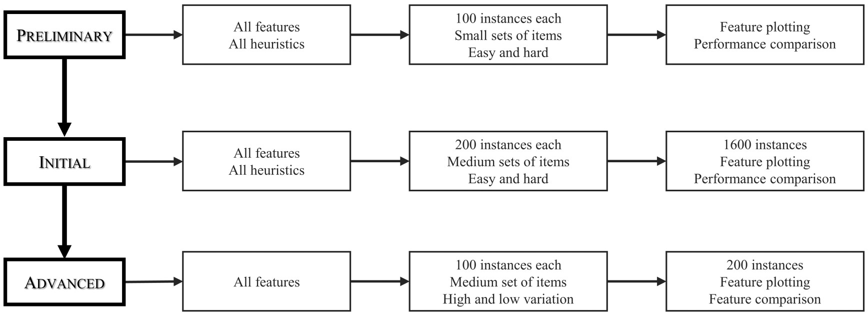


Fig. 3. Methodology followed throughout this work.

4.1 Preliminary Testing

Throughout this first stage, we focus on exploring whether our idea is promising. Thus, we strive to generate both, easy and hard instances, for each of the heuristics discussed in Section 2.2. For this batch of tests, we created 100 instances with 50 items each, for every scenario. This represents a total of 800 instances (8×100). In the first case, we used equation (1) to maximize the difference between the fitness (i.e., the average waste) given by the target heuristic (Fit_{one}) and that of the best remaining heuristic ($\min(Fit_{others})$). In the second case, we used equation (2) to also maximize the distance between the performance of the target heuristic and the worst of the remaining heuristics ($\max(Fit_{others})$).

$$F_{obj} = -(\min(Fit_{others}) - Fit_{one}) \tag{1}$$

$$F_{obj} = -(Fit_{one} - \max(Fit_{others})) \tag{2}$$

4.2 Initial Testing

Afterwards, we push our idea a bit further and strive to generate bigger instances. Hence, we demand sets of 100 items. Once again, we create instances for all 8 scenarios (i.e., 4 heuristics and 2 conditions). Moreover, we use our approach to generate 200 instances per scenario, to analyze the repeatability of our idea. This leads to a total of 1600 instances during this testing stage.

4.3 Advanced Testing

As a final effort to test our approach, we concentrate on generating a different kind of instance. This time, we modify the objective function in order to evolve instances with a maximum difference among solvers, and instances where all solvers perform the same. In the first case, we use equation (3) to maximize the standard deviation that heuristics exhibit over the instance. Here, Fit_i is the fitness (i.e., the average waste) of every heuristic, Fit_{avg} is the average fitness achieved by all solvers, and N_H is the number of heuristics (i.e., four for this work).

$$F_{obj} = -\sqrt{\frac{\sum\limits_{i=1}^{N_H} (Fit_i - Fit_{avg})^2}{N_H - 1}} \tag{3}$$

In the second one, we use equation (4) to search for instances where all solvers perform equally. Thus, Fit_{Best} is the best fitness achieved by all solvers, whilst Fit_{Worst} is the worst one.

$$F_{obj} = (Fit_{Best} - Fit_{Worst})^2 \tag{4}$$

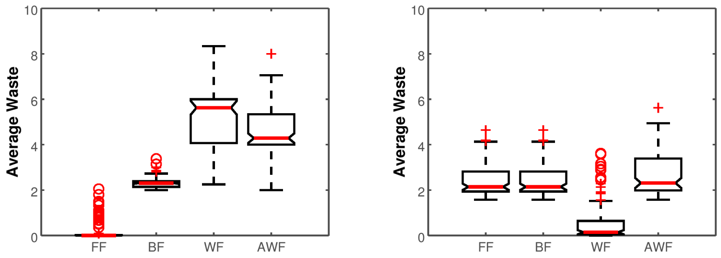
We pursue this idea to try and identify, based on the features from Section 2.3, the regions in the feature space where it is critical to select an appropriate solver and regions where it is of no importance. Since the idea is to also explore the versatility of our model, during this stage we use instances with 100 items each, and create 100 instances for each of the two scenarios.

5 Experiments and Results

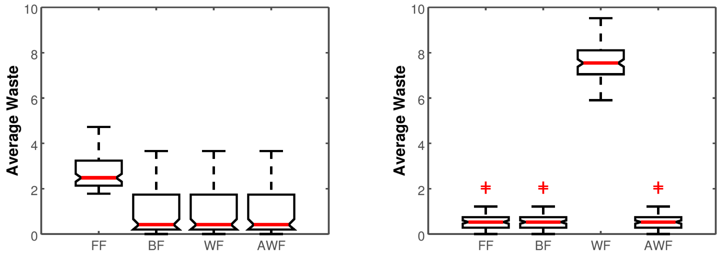
This section presents the most relevant data of our experiments. Because of space restrictions, we only focus on showing the performance of some heuristics in terms of average waste. Nonetheless, this does not mean that the other metrics were unsatisfactory.

5.1 Preliminary Testing

Figure 4 summarizes the average waste of the BP-FF and BP-WF heuristics over all generated instances. The first row relates to instances that can be efficiently solved by one heuristic (e.g., BP-FF or BP-WF), but which are poorly solved by the remaining ones. Similarly, the second row relates to instances poorly solved by one heuristic and efficiently solved by the remaining ones. In both cases, our proposed approach successfully tailors instances. In the first case, the



(a) FF (Easy) (b) WF (Easy)



(c) FF (Hard) (d) WF (Hard)

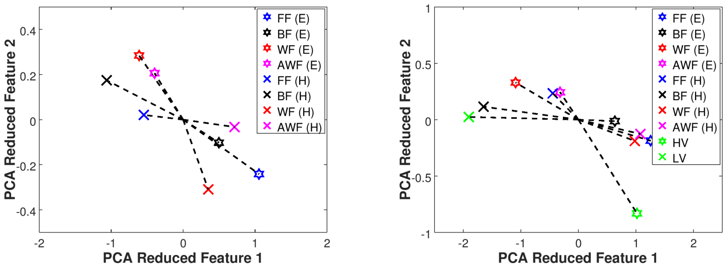
Fig. 4. Average waste achieved by the First Fit (FF) and the Worst Fit (WF) heuristics over the sets of instances generated in the preliminary testing stage. Boxplots reflect data for 100 instances. Other solvers are omitted due to space restrictions.

average waste becomes minimum. In the second one, the average waste increases (sometimes even dramatically, e.g. for BP-WF).

Figure 5(a) shows the center of each set of instances, located at different positions. Interestingly, easy and hard to solve instances, for a specific heuristic, exhibit opposing behaviors. For example, increasing the difficulty of instances for BP-FF, leads to instances whose items are more varied, but which are also two units longer (on average). Even more, there are 10% more big pieces within hard instances. Nonetheless, in the case of BP-WF harder instances are those with fewer big pieces (about 7% less), leading to smaller items (about one unit on average).

5.2 Initial Testing

Although increasing the number of items in the instance to 100 makes it harder for our model to generate the instances with the requested behavior, it is still able to generate quite useful results (Fig. 6). This time around, there are also scenarios where a solver has virtually no waste at all (e.g., for the best fit heuristic), and scenarios where a heuristic wastes way more space than other approaches. Even in those cases where the behavior of the heuristic of interest is not that different



(a) Preliminary stage (b) Initial stage

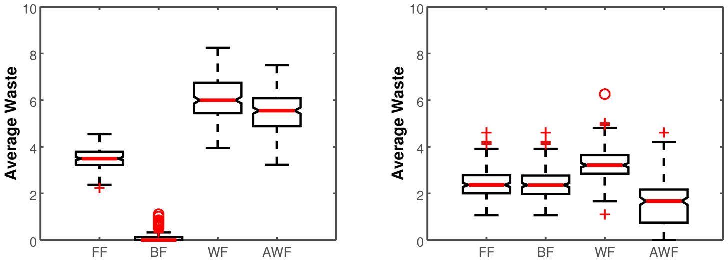
Fig. 5. Centroid of all instances generated in the preliminary and initial testing stages. Stars indicate easy-to-solve (E) instances for a specific heuristic and crosses indicate hard-to-solve (H) instances for a specific heuristic. Data is also shown for instances with High variation (HV) and Low variation (LV).

from the others, there are still elements worth remarking. For example, consider hard-to-solve instances for the best fit heuristic. Even though medians are quite close, best fit is the only heuristic unable to achieve average waste values below one.

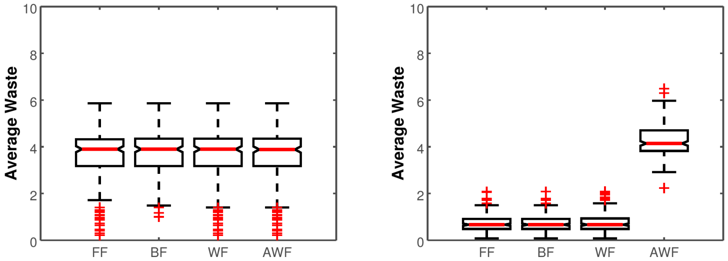
A plot of the location of the 1600 generated instances, again reveals an interesting pattern (Fig. 5(b)). As in the previous stage, easy and hard instances are located on opposing parts of the feature space. It is also interesting to observe that changing the difficulty of a heuristic implies relatively the same movement in the feature space: for the first fit and best fit heuristics, it represents a shift from right to left and upwards; for the worst fit and almost worst fit, it changes and now goes from left to right and downwards. Hence, the behaviour from the previous stage holds. In fact, the ratio of big pieces (RBP) increased from 39% to 51% for BP-FF, while it decreased from 55% to 41% for BP-WF.

5.3 Advanced Testing

As mentioned in Section 3, this stage pushes our approach further by trying out different objective functions. Figure 7 shows two scenarios of interest: one with varied performance, and one with the same performance. Once again, our proposed model was able to cope with the situation, evolving instances favorable for the user needs. Taking into account that the bin capacity was set to 30, we consider that a gap in heuristic performance of about 10 is quite remarkable. Moreover, the model was actually able to evolve instances where heuristics had the exact same behavior, thus representing a region of the feature space where it is irrelevant to spend resources on determining which heuristic to use. The centroid of these two sets of instances can be seen in Fig. 5(b). As expected, the scenario with the highest variation is the one farthest away. This can be explained since the other scenarios only considered that a given solver was either good or bad in comparison to the others, but did not seek that the solvers exhibited



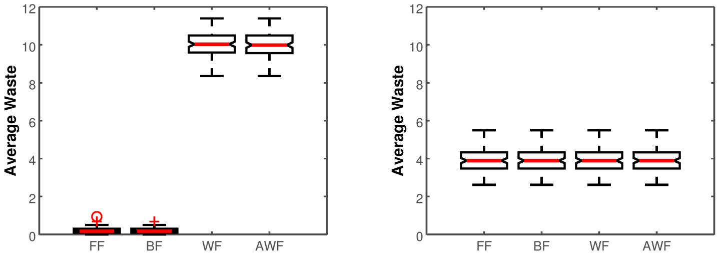
(a) Best Fit (Easy) (b) Almost Worst Fit (Easy)



(c) Best Fit (Hard) (d) Almost Worst Fit (Hard)

Fig. 6. Average waste achieved by the Best Fit (BF) and the Almost Worst Fit (AWF) heuristics over all the sets of instances generated in the initial testing stage. Boxplots reflect data for 200 instances. Other solvers are omitted due to space restrictions.

a varied performance. Similarly, the set of instances with the lowest variation is close to the set of hard-to-solve instances for the best fit heuristic. This can also be explained since, as it was mentioned above, this scenario exhibits a low variation (with only some instances yielding average waste values below unity).



(a) High Variation (b) Low Variation

Fig. 7. Average waste achieved by all solvers when operating over instances generated with high variation (a) and with low variation (b).

6 Conclusions

In this work we presented an evolutionary-based model for tailoring instances to specific needs. The tool we propose may allow for an in-depth study of different heuristics. It can be used to create instances that exploit heuristic strengths and weaknesses, so knowledge about when to use each one may be derived. However, we did not pursue such endeavor here, mainly due to space restrictions. Nonetheless, we tested our approach by generating over 2500 instances for the 1D Bin Packing Problem, distributed along four different heuristics. Our testing included scenarios where one solver excels while the others fail, and scenarios where one solver fails while the remaining ones excel. To push our generation model even further, we included instances where the solvers performed as diversely as possible, and instances where all solvers performed equally.

Data exhibited interesting elements. For example, we created easy instances for the first fit heuristic (BP-FF), where the median number of bins was between 7% and 20% lower than for the remaining heuristics. Similarly, the median number of completely filled bins for BP-FF was over 10% higher than for the second best heuristic, and about eight times higher than for the worst heuristic. In some cases results were not as astonishing. But, there was always some benefit. Consider easy-to-solve instances for the worst fit heuristic, and hard-to-solve instances for the best fit heuristic. In the former, the heuristic allowed for near zero average waste. In the latter, the heuristic was the only one unable to achieve near zero average waste. This means that, through our approach, one can find instances with specific behaviors without the need for exhausting all combinations of elements, nor deriving mathematical expressions that relate the items.

Also worth noting is that as difficulty increases, instance location within the feature domain shifts. For example, for the first fit and best fit heuristics, instances shifted from right to left and upwards. However, for the worst fit and almost worst fit, they migrated from left to right and downwards.

Regarding the final batch of tests, we can conclude that our proposed model can adapt to different kinds of situations. In this case, it was able to evolve instances where solvers exhibited a high variation on their performance, generating gaps of average waste of about 33%. Moreover, it was also able find configurations where all solvers performed exactly the same, representing the region of the feature space where it becomes unnecessary to select a specific heuristic.

References

1. Beasley, J.: OR-library: Distributing test problems by electronic mail. The Journal of the Operational Research Society **41**(11), 1069–1072 (1990)
2. Drake, J.H., Swan, J., Neumann, G., Özcan, E.: Sparse, Continuous Policy Representations for Uniform Online Bin Packing via Regression of Interpolants. In: Evolutionary Computation in Combinatorial Optimization. EvoCOP 2017. Lecture Notes in Computer Science, pp. 189–200. Springer (2017). https://doi.org/10.1007/978-3-319-55453-2_13

3. Gomez, J.C., Terashima-Marín, H.: Evolutionary hyper-heuristics for tackling bi-objective 2D bin packing problems. Genetic Programming and Evolvable Machines (mar 2017). https://doi.org/10.1007/s10710-017-9301-4
4. van Hemert, J.I.: Evolving binary constraint satisfaction problem instances that are difficult to solve. In: Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC'03). pp. 1267–1273. IEEE Press (2003)
5. van Hemert, J.I.: Evolving combinatorial problem instances that are difficult to solve. Evolutionary Computation **14**(4), 433–462 (2006)
6. Knowles, J.D., Corne, D.W.: Approximating the nondominated front using the pareto archived evolution strategy. Evolutionary computation **8**(2), 149–172 (2000)
7. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Mathematical Programming Computation **3**(2), 103–163 (2011)
8. López-Camacho, E., Terashima-Marín, H., Ross, P.: A Hyper-heuristic for Solving One and Two-dimensional Bin Packing Problems. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'11) pp. 257–258 (2011). https://doi.org/10.1145/2001858.2002003
9. Lust, T., Teghem, J.: The multiobjective multidimensional knapsack problem: A survey and a new approach. International Transactions in Operational Research **19**(4), 495–520 (2012)
10. Martello, S., Pisinger, D., Vigo, D.: The three-dimensional bin packing problem. Operations Research **48**(2), 256–267 (2000)
11. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. John Wiley & Sons (1990)
12. Özcan, E., Parkes, A.J.: Policy matrix evolution for generation of heuristics. Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11 p. 2011 (2011). https://doi.org/10.1145/2001576.2001846
13. Petursson, K.B., Runarsson, T.P.: An evolutionary approach to the discovery of hybrid branching rules for mixed integer solvers. Proceedings - 2015 IEEE Symposium Series on Computational Intelligence, SSCI 2015 pp. 1436–1443 (2016)
14. Pisinger, D.: Where are the hard knapsack problems? Computers & Operations Research **32**(9), 2271–2284 (2005)
15. Smith-Miles, K., van Hemert, J.: Discovering the suitability of optimisation algorithms by learning from evolved instances. Annals of Mathematics and Artificial Intelligence **61**(2), 87–104 (2011)
16. Smith-Miles, K., van Hemert, J., Lim, X.: Understanding tsp difficulty by learning from evolved instances. In: Blum, C., Battiti, R. (eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science, vol. 6073, pp. 266–280. Springer Berlin Heidelberg (2010)
17. Sosa-Ascencio, A., Terashima-Marín, H., Ortiz-Bayliss, J.C., Conant-Pablos, S.E.: Grammar-based Selection Hyper-heuristics for Solving Irregular Bin Packing Problems. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO '16 Companion. pp. 111–112. ACM Press, New York, New York, USA (2016). https://doi.org/10.1145/2908961.2908970
18. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems pp. 95–100 (2001)
19. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation **3**(4), 257–271 (1999)