Using μ Genetic Algorithms for Hyper-heuristic Development: A Preliminary Study on Bin Packing Problems

José C. Ortiz-Bayliss, Julio Juárez, Jesús Guillermo Falcón-Cardona
Tecnologico de Monterrey, School of Engineering and Sciences, Monterrey 64849, Mexico,
E-mails: {jcobayliss, juarez.julio, jfalcon}@tec.mx

***Abstract*—Hyper-heuristics have gained attention from the community in the last decade, given their capability to deal with complex combinatorial problems. Although they do not guarantee optimality, they provide a mechanism to benefit from the strengths of different heuristics and obtain good-quality solutions. So, by selectively applying such heuristics, the search is optimized. Although evolutionary algorithms have successfully been used to generate hyper-heuristics for various problems, the number of objective function evaluations is always an issue. We propose using a μ Genetic Algorithm to address this severe criticism when producing hyper-heuristics. We tested our approach on the one-dimensional Bin Packing Problem, and the results suggest that this approach is competitive concerning the heuristics applied in isolation.**

***Index Terms*—Micro Genetic Algorithms, Hyper-heuristics, Bin Packing Problem**

I. Introduction

The Bin Packing Problem (BPP) appears frequently in its different variants [8], [14]. For example, in our daily lives, we may think of the problem of choosing which clothes to pack in our luggage in preparation for a flight. We know that we must use a limited number of suitcases to pack everything, or we will be forced to pay an extra fee. In general, BPP interests the community since many other optimization problems can be modeled as BPPs [13], [16].

Like other optimization problems, the BPP exhibits a complexity that grows exponentially with the input size. Hence, the importance of methods that adequately deal with such a vast search space. The specialized literature contains many methods to solve the BPP [1], [4], [25]. Some of them are tailor-made and specific to particular variants and conditions. Although limited in application, those methods usually provide state-of-the-art results. Conversely, other methods aim for generality, achieving an 'acceptable' performance but knowing that they will hardly produce the optimal result. Most of these methods rely on approximation methods such as heuristics and metaheuristics.

Hyper-heuristics represent an alternative to deal with the complexity of combinatorial optimization problems [6]. Instead of using metaheuristics to solve the problem directly, we can use metaheuristics to produce solvers that combine the strengths of other 'simple' methods (in this case, heuristics). We usually refer to such methods as "low-level solvers". Using rules, we decide when to apply each heuristic based on the problem state under exploration. This type of hyper-heuristic is usually referred to as "selection hyper-heuristic" [15]. Although other types of hyper-heuristics exist (such as sequence-based [27], batched-mode [3], and generation hyper-heuristics [17], to mention a few), we will not delve into their behavior since they are beyond the scope of this work. However, some of these types of hyper-heuristics may appear in the document as part of the literature review.

This work explores using a canonical Micro Genetic Algorithm (μGAs) to produce hyper-heuristics. A μGA is a variation of the standard general genetic algorithm in which the population is smaller (only a few individuals), and the genetic operators omit mutation [10]. To avoid stagnation, the μGA uses a restart method on the population to keep the best solution found and reset the others. We use one widespread combinatorial optimization problem, the one-dimensional bin packing problem, to evaluate the performance of the hyper-heuristics produced with the μGA. In this problem, the task is to pack a set of one-dimensional items into one-dimensional bins. Solving this problem requires minimizing the number of bins used to pack all the items. Although we chose this problem for convenience, the hyper-heuristic model proposed in this work can be applied to other problem domains with minimum changes.

The remainder of this document is as follows. Section II provides a context of our work by providing relevant concepts and a short review of related works. In Sect. III, we provide details of our solution model and how we propose to create and test hyper-heuristics with μGAs for solving the BPP. Section IV describes the experiments conducted and analyses their main results. Finally, Sect. V presents the conclusion and provides potential paths for future work.

II. Background

A. Problem definition

Bin packing is a classic optimization problem in computer science [20]. This problem aims to allocate items of various sizes into the minimum number of bins without exceeding their capacity. A formal definition of the problem is as follows. Given a finite set of items I, a size s_i associated with each item $i \in I$ and a bin capacity B,

$$\text{Minimize} \quad \sum_{i \in I} y_j \tag{1}$$

subject to,

$$\sum_{i \in I} x_{ij} s_i \leq B y_j \qquad \forall j \in \{1, \cdots, n\}, \tag{2}$$

$$\sum_{j}^{n} x_{ij} = 1 \qquad \forall i \in I, \tag{3}$$

$$x_{ij} \in \{0,1\}, \quad y_j \in \{0,1\}, \tag{4}$$

where $x_{ij} = 1$ if item i is allocated in bin j, $x_{ij} = 0$ otherwise, and $y_j = 1$ if bin j is used, $y_j = 0$ otherwise. Note that the constraints indicate that bins cannot exceed their capacity, and each item must be allocated to at most one bin.

This problem is NP-hard, and its corresponding decision version is NP-Complete, which means that computing an optimal solution in a reasonable time is out of the reach of current algorithms [20]. This calls for alternative approaches that focus on producing near-optimal solutions in polynomial time. One alternative aims to solve the problem through high-level optimization approaches such as hyper-heuristics.

B. Related work

The overlap between hyper-heuristics and evolutionary algorithms has been a prominent topic in hyper-heuristic literature. For example, one of the first works on hyper-heuristics — although it did not mention such a term— goes back to 1999. That work involved Genetic Algorithms (GAs) for solving the Examination Timetabling Problem [29]. Subsequent works by Cowling et al. used GAs to produce hyper-heuristics for various scheduling and packing problems [11], [12]. Genetic Programming (GP) was first used to produce hyper-heuristics in 2006 by Burke et al., where they solved the bin packing problem [7]. Various researchers further extended the idea of using GP to produce hyper-heuristics to cover other problems and scenarios in subsequent years [2], [5]. In 2006, we observed the first work on messy GAs to produce hyper-heuristics for the 2D-regular Cutting Stock Problem by Terahsima et al. [28]. They continued exploring this approach for the 2D Bin Packing Problem [21] and Constraint Satisfaction Problems [22].

Although we lack the space for a comprehensive literature review of all hyper-heuristic works that have involved evolutionary computation, we would like to take a few lines to stress the most recent developments, which focus mainly on multi-objective applications [18], [19], and solving scheduling problem variations [9], [17], [26].

To the authors' knowledge, no previous work explores using μGAs to produce hyper-heuristics. For this reason, in this work, we propose using such a technique and benefit from the idea of the small populations to reduce the number of objective function evaluations without sacrificing the performance shown by other traditional evolutionary algorithms such as GAs.

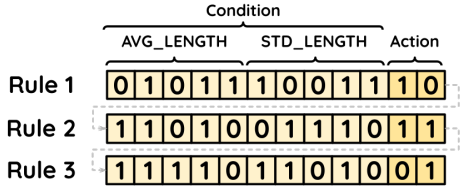


Fig. 1: An example of a chromosome that codes three *condition→action* rules.

III. Solution Approach

Our proposed approach uses a μGA that evolves a population of selection constructive hyper-heuristics for the BPP. Before describing how the μGA evolves the hyper-heuristics, it is mandatory to describe the codification of a hyper-heuristic into a chromosome. Once we state the codification of a hyper-heuristic (and, thus, the representation of the search space), we can describe how the μGA searches for approximately good hyper-heuristics.

A. Chromosome codification

It is worth noting that the search space of the μGA is the space of all possible hyper-heuristics that solve the BPP. Each hyper-heuristic is characterized as a set of *condition→action* rules for constructing solutions step by step. For each rule, a *condition* is associated with a problem state, and an *action* represents a heuristic.

Since we focus on a one-dimensional BPP, we can only use the length of the items for characterizing the instances. Thus, a rule's *condition* characterizes a BPP instance. We utilize two popular features related to the lengths of the items within the instance:

- **AVG_LENGTH**: the average length of the items within the instance, and
- **STD_LENGTH**: the standard deviation of the length of the items in the instance.

For better performance, these values are normalized by dividing the average and standard deviation by the maximum length among the items considered (in this work, it is 32). We should mention that other features might be used (e.g., the proportion of 'large' and 'small' items). However, we decided to use AVG_LENGTH and STD_LENGTH to keep the characterization as simple as possible.

The *action* of a rule represents a heuristic to solve the BPP. We have considered four commonly used heuristics for the BPP [24]. These heuristics are briefly described below.

- **First Fit (FF)**. FF packs the item in the first bin with enough capacity for packing the item.
- **Best Fit (BF)**. BF packs the item in the bin that minimizes the waste.
- **Worst Fit (WF)**. WF packs the item in the bin that maximizes the waste.
- **Almost Worst Fit (AWF)**. AWF applies WF twice, allowing it to pack the item in the bin with the second most waste produced.

If no bin is available to pack the current item, the system opens a new bin to pack such an item. In the event of ties, the first bin that meets the heuristic's criterion is preferred.

At this point, we are in the position to describe how to code a hyper-heuristic. Figure 1 shows a chromosome that codes three *condition→action* rules. In general, a *condition→action* rule is represented as a binary string of length $mk + \log_2(n)$, where m is the number of features of the condition, k is the number of bits used to code each feature in the range $[0, 1]$, and n is the number of available heuristics to choose from. In Figure 1, $m = 2$ because we use AVG_LENGTH and STD_LENGTH as features, $k = 5$, and $n = 4$ since we defined four heuristics, namely, FF, BF, WF, and AWF whose binary codes are 00, 01, 10, and 11, respectively. It is worth noting that 2^k different values can be represented for each coded feature. When decoding the value of a feature, the corresponding 10-base value is calculated as $\frac{\sum_{j=0}^{k-1} 2^j \cdot I_j}{2^k - 1}$, where $I_j = 1$ if the j^{th} bit is equal to 1, or $I_j = 0$, otherwise. For example, AVG_LENGTH $= 01011$ and STD_LENGTH = 10011, in rule 1 of the chromosome in Figure 1, and their 10-base value are $11/31 \approx 0.3548$ and $19/31 \approx 0.6129$. Consequently, the complete rule is decoded as $(0.3548, 0.6129, WF)$.

Thus, using the AVG_LENGTH and STD_LENGTH we can characterize both the problem state and the conditions that trigger an action. For example, say the problem state at moment t is $s_t = (\text{AVG\_LENGTH} = 0.9123, \text{STD\_LENGTH} = 0.8176)$ and the decoded condition-action rules of the HH (from Figure 1) are $r_1 = (0.3548, 0.6129, \text{WF})$, $r_2 = (0.8387, 0.04516, \text{AWF})$ and $r_3 = (0.9677, 0.8387, \text{FF})$. Then, using Euclidean distance, we determine the closest condition to the problem state, which corresponds to r_3. Finally, r_3 triggers heuristic FF, which in turn updates the problem state to s_{t+1}, and the whole process is repeated until there are no more items left to allocate. Figure 3 shows a diagram of the process on how the HH solves a BPP instance.

Since the approach is evolving problem solvers for the BPP, the fitness value of chromosomes is expected to represent how well its encoded HH generalizes the problem. In this context, the average bin usage (ABU) is both an indicator of HH performance and a fitness value for the μGA. The usage of a single bin is calculated as the sum of the lengths of the items contained in the bin divided by the bin's capacity. Thus, the ABU is the average of the usage among all bins. The closer the ABU is to 1.0, the smaller the waste of space per bin, the fewer bins needed, and the better the solution. In this context, the ideal heuristic will always produce solutions that yield an ABU of 1.0.

B. Operation of the μGA

Figure 2 shows the overall sequence of actions followed to produce and test a hyper-heuristic using a μGA. The process starts by creating a small random population of hyper-heuristics represented by N chromosomes. Each chromosome is randomly initialized with three to five coded *condition→action* rules. Then, each chromosome is evaluated over all the instances of the training set to compute its ABU fitness value.

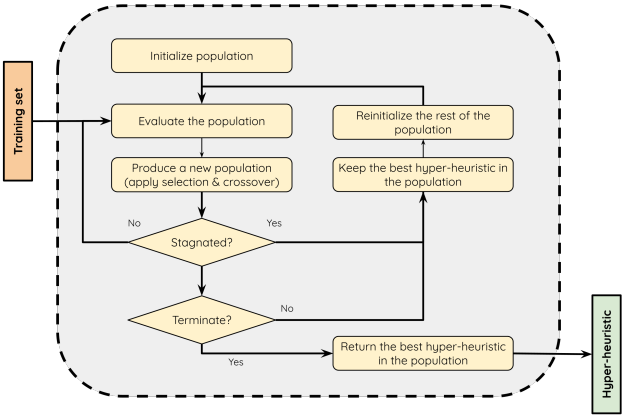


Fig. 2: A graphical depiction of the μGA that searches for hyper-heuristics to solve the one-dimensional BPP.

Once the initial population is evaluated, we use binary tournament selection and one-point crossover to produce an offspring population. Due to the one-point crossover, offspring solutions might have chromosomes of different sizes than their parents. In other words, our μGA uses variable-length chromosomes. It is worth emphasizing that the crossover point is calculated on the parent with the smallest chromosome. Once the offspring population is generated, it replaces the previous one. If the population is stagnated (there is little difference between the fitness of the hyper-heuristics in the population) and the termination criterion (number of evaluations) has not been reached, a new randomly generated population of hyper-heuristics replaces the previous population (all but the most fitted hyper-heuristic). Then, the process is repeated from the population evaluation. When the termination criterion is reached, the best hyper-heuristic in the last population is returned as the hyper-heuristic produced by the model. Finally, we solve the instances in the test set using the hyper-heuristic the μGA produced.

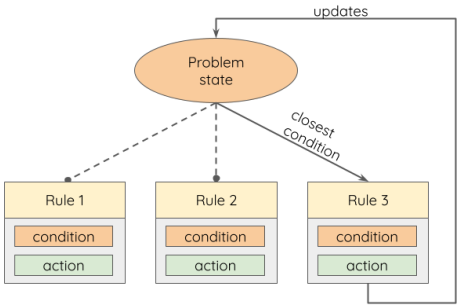


Fig. 3: Example of the process how the hyper-heuristic solves a BPP instance.

IV. Experiments and Results

This work explores using μGAs to produce selection constructive hyper-heuristics (HHs) for the BPP. To test our

proposal, we conducted two rounds of experiments. The first round tests our method on particular pathological instances. In the second round, we performed a more general performance test. In all cases, for the μGA, we used populations of five individuals, with a crossover rate of one and a mutation rate of zero, running for 300 objective function evaluations.

For both rounds of experiments, we used synthetic sets of BPP instances obtained through an evolutionary-based generator of synthetic instances proposed by Plata et al. [23]. This method can generate instances of knapsack-type problems tailored to favor or hinder the performance of particular heuristics. We generated four sets of 125 pathological instances specific to each of the four heuristics FF, BF, WF, and AWF (500 instances in total). We call these generated instance sets I_{FF}, I_{BF}, I_{WF}, I_{AWF}. Furthermore, each set was split into two subsets, using 25 instances for training and the remaining 100 for testing.

A. Experiment I

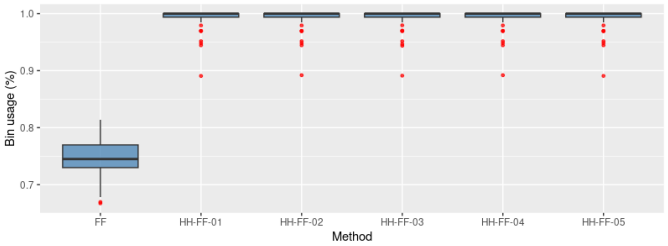
Hyper-heuristics are particularly useful when dealing with new problems for which our understanding is limited or with pathological instances of available algorithms. In this spirit, using our μGA approach, we generate five HHs for each of the pathological instance training sets. That is, five HHs were evolved using the training set I_{FF}, another five HHs using the I_{BF}, and so on. To name these hyper-heuristics, we used the prefix "HH-", followed by the name of the heuristic affected in the training set. Finally, we added the run number as a postfix to identify each hyper-heuristic. For example, HH-WF-03, represents the third hyper-heuristic produced using the training instances from I_{WF}.

The results shown in Figure 4(a)–(d) demonstrate that the hyper-heuristics have learned how to combine the heuristics in such a way that their performance is as expected: they overcome heuristics that are not suitable for a specific instance set. These results also confirm the hypothesis that the μGA can produce competent solvers that learn to combine heuristics to overcome their drawbacks.

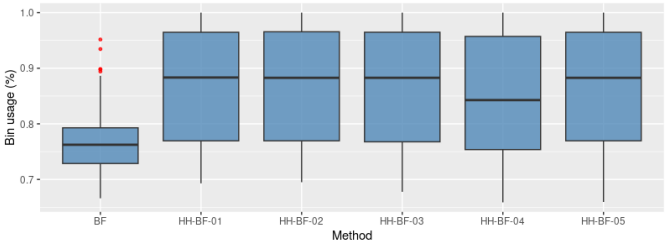
B. Experiment II

However, a more interesting and challenging task appears when we combine the strengths of the four heuristics. In this experiment, we merged the four training sets into a single one and used it to produce five hyper-heuristics (HH-01 to HH-05). Once produced, we applied such methods to the test set and registered their results (in terms of ABU). The behavior of these new hyper-heuristics and the four heuristics on the test set is depicted in Figure 5.

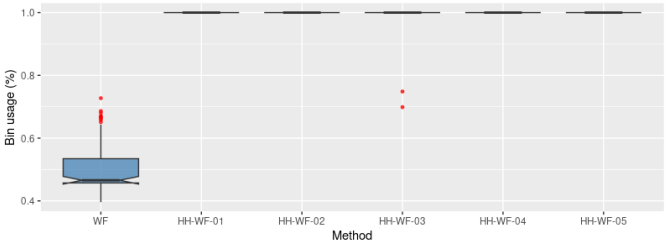
We observed through this experiment that the hyper-heuristics somehow replicate the behavior of BF, the best individual performer on the training set. In this case, BF was also the best performer on the test set, with 93.97% of ABU. For this reason, the five hyper-heuristics easily overcome FF, WF, and AWF, with 90.25%, 83.77%, and 89.92% values for ABU, respectively. We can rank HH-01 to HH-05 according to their performance in terms of ABU as follows. The lowest



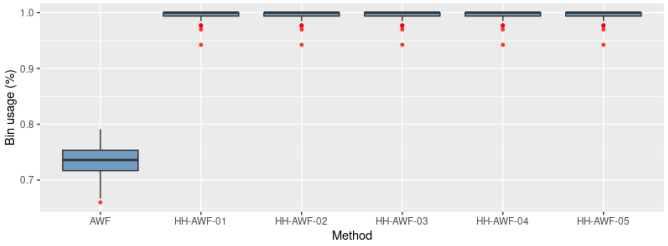
(a) FF



(b) BF



(c) WF



(d) AWF

Fig. 4: Performance (in terms of ABU) of the four heuristics and the hyper-heuristics trained to solve the test instances from I_{FF}, I_{BF}, I_{WF}, and I_{AWF}.

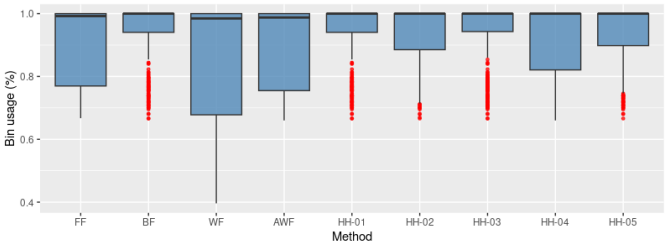


Fig. 5: Performance (in terms of ABU) of the four heuristics and the hyper-heuristics trained to solve the complete test set.

ABU was obtained by HH-04, with 92.92. HH-02 and HH-03 continue the list with 93.63% and 93.87%, respectively. HH-01 is the first hyper-heuristic that rivals BF. In fact, HH-01 replicates BF; they always make the same decisions. Finally, HH-03 performed beyond expectations, obtaining an ABU of 94.06%, improving not only on FF, WF, and AWF (as the rest of the hyper-heuristics) but also on BF, the best individual performer. Although the improvement of HH-03 over BF seems marginal (0.09% in ABU), it suggests that it is indeed feasible to improve the performance of individual heuristics through hyper-heuristics produced with μGAs, as it is proposed in this work.

V. Conclusion

Based on our results, we consider that, on an individual level, it is indeed possible to produce hyper-heuristics —by means of the μGA— that rival some of its heuristic components. However, in this work, BF was the best-performing heuristic in almost all the cases. This behavior makes it challenging to produce a hyper-heuristic that does not include BF as one of the options. Since BF is the best option for many instances, the hyper-heuristics tend to replicate its behavior. This phenomenon is similar to what happens in other learning approaches when the training set is biased. Although this does not mean a bad result since the hyper-heuristics do learn, it opens a path for future work in which we should explore ways to overcome the problem of working with 'unbalanced' sets of instances, where one of the heuristics is almost always the best performer.

References

[1] M. Abdel-Basset, G. Manogaran, L. Abdel-Fatah, and S. Mirjalili. An improved nature inspired meta-heuristic algorithm for 1-D bin packing problems. *Personal and Ubiquitous Computing*, 22(5-6):1117–1132, Oct 2018.

[2] S. Allen, E. K. Burke, M. Hyde, and G. Kendall. Evolving reusable 3d packing heuristics with genetic programming. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, page 931–938, New York, NY, USA, 2009. ACM.

[3] S. Asta, E. Özcan, and A. J. Parkes. Batched mode hyper-heuristics. In G. Nicosia and P. Pardalos, editors, *Learning and Intelligent Optimization*, pages 404–409, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] S. Asta, E. Özcan, and A. J. Parkes. CHAMP: Creating heuristics via many parameters for online bin packing. *Expert Systems with Applications*, 63:208–221, Nov 2016.

[5] M. Bader-El-Den and R. Poli. Generating sat local-search heuristics using a gp hyper-heuristic framework. In N. Monmarché, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, editors, *Artificial Evolution*, pages 37–49, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[6] E. K. Burke, M. Gendreau, M. R. 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*, 64:1695–1724, 2013.

[7] E. K. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervós, L. D. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, pages 860–869, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] S. Cai, J. Deng, L. H. Lee, E. P. Chew, and H. Li. Heuristics for the two-dimensional irregular bin packing problem with limited rotations. *Computers & Operations Research*, 160, 2023.

[9] X. Chen, G. Jiang, Y. Xiao, G. Li, and F. Xiang. A hyper heuristic algorithm based genetic programming for steel production scheduling of cyber-physical system-oriented. *Mathematics*, 9(18), 2021.

[10] C. A. Coello Coello Coello and G. Toscano Pulido. A micro-genetic algorithm for multiobjective optimization. In E. Zitzler, L. Thiele, K. Deb, C. A. Coello Coello, and D. Corne, editors, *Evolutionary Multi-Criterion Optimization*, pages 126–140, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[11] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, pages 176–190, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[12] P. I. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *4th Metaheuristics International Conference (MIC)*, 2002.

[13] M. Delorme, M. Iori, and S. Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. In *EJOR*, volume 255, pages 1–20. Elsevier, Nov 2016.

[14] M. Delorme, M. Iori, and S. Martello. Bpplib: a library for bin packing and cutting stock problems. *Optimization Letters*, 12(2):235–250, 2018.

[15] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke. Recent advances in selection hyper-heuristics. *EJOR*, 285(2):405–428, 2020.

[16] U. Eliiyi and D. T. Eliiyi. Applications of bin packing models through the supply chain. *International journal of business and management*, 1(1):11–19, Jun 2009.

[17] H. Fan, H. Xiong, and M. Goh. Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints. *Computers & Operations Research*, 134:105401, 2021.

[18] G. Guizzo, S. R. Vergilio, A. T. Pozo, and G. M. Fritsche. A multi-objective and evolutionary hyper-heuristic applied to the integration and test order problem. *Applied Soft Computing*, 56:331–344, 2017.

[19] J. Heise and S. Mostaghim. Online learning hyper-heuristics in multi-objective evolutionary algorithms. In *Evolutionary Multi-Criterion Optimization: 12th International Conference, EMO 2023, Leiden, The Netherlands, March 20–24, 2023, Proceedings*, page 162–175, Berlin, Heidelberg, 2023. Springer-Verlag.

[20] B. H. Korte, J. Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[21] E. López-Camacho, H. Terashima-Marin, P. Ross, and G. Ochoa. A unified hyper-heuristic framework for solving bin packing problems. *Expert Systems with Applications*, 41(15):6876–6889, 2014.

[22] J. C. Ortiz-Bayliss, H. Terashima-Marín, and S. E. Conant-Pablos. Combine and conquer: An evolutionary hyper-heuristic approach for solving constraint satisfaction problems. *Artif. Intell. Rev.*, 46(3):327–349, oct 2016.

[23] L. F. Plata-González, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. Coello Coello. Evolutionary-based tailoring of synthetic instances for the knapsack problem. *Soft Computing*, 23(23):12711–12728, 2019.

[24] A. Silva-Gálvez, J. Orozco-Sanchez, E. Lara-Cárdenas, J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, and H. Terashima-Marín. Discovering action regions for solving the bin packing problem through hyper-heuristics. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 822–828, 2020.

[25] K. Sim, E. Hart, and B. Paechter. A Lifelong Learning Hyper-heuristic Method for Bin Packing. *Evolutionary Computation*, 23(1):37–67, Mar 2015.

[26] H.-B. Song and J. Lin. A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times. *Swarm and Evolutionary Computation*, 60:100807, 2021.

[27] X. Sánchez-Díaz, J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, S. E. Conant-Pablos, and H. Terashima-Marín. A feature-independent hyper-heuristic approach for solving the knapsack problem. *Applied Sciences*, 11(21), 2021.

[28] H. Terashima-Marín, C. J. Farías Zárate, P. Ross, and M. Valenzuela-Rendón. A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, page 591–598, New York, NY, USA, 2006. ACM.

[29] H. Terashima-Marín, P. Ross, and M. Valenzuela-Rendón. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, GECCO'99, page 635–642, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.