A Preliminary Study on Score-based Hyper-heuristics for Solving the Bin Packing Problem

A. Silva-Gálvez$^{[0000-0003-0394-5555]}$, E. Lara-Cárdenas$^{[0000-0002-6357-4680]}$, I. Amaya$^{[0000-0002-8821-7137]}$, J. M. Cruz-Duarte$^{[0000-0003-4494-7864]}$, and J. C. Ortiz-Bayliss$^{[0000-0003-3408-2166]}$

School of Engineering and Sciences, Tecnologico de Monterrey
Av. Eugenio Garza Sada 2501, Monterrey, NL 64849, Mexico
artsg130994@gmail.com, a00398510@itesm.mx,
{iamaya2, jorge.cruz, jcobayliss}@tec.mx

Abstract. The bin packing problem is a widespread combinatorial problem. It aims at packing a set of items by using as few bins as possible. Among the many available solving methods, approximation ones such as heuristics have become popular due to their reduced cost and generally acceptable solutions. A further step in this regard is given by hyper-heuristics, which literature usually defines as "high-level heuristics to choose heuristics". Hyper-heuristics choose one suitable heuristic from a set of available ones, to solve a particular portion of an instance. As the search progresses, heuristics can be exchanged, adapting the solution process to the current problem state under exploration. In this work, we describe how to generate and use hyper-heuristics that keep a record of the scores achieved by individual heuristics on previously solved bin packing problem instances in the form of rules. Then, hyper-heuristics manage those scores to estimate the performance of such heuristics on unseen instances. In this way, the previous actions of the hyper-heuristics determine which heuristic to use on future unseen cases. The experiments conducted under different scenarios yield promising results where some of the hyper-heuristics produced outperform isolated heuristics.

Keywords: Bin packing problem · Heuristic · Hyper-heuristic.

1 Introduction

The bin packing problem (BPP) [8, 14], in its general formulation, consists of packing a set of items (with their corresponding properties) by minimizing the number of bins used. In general, BPP is an exciting problem since many other optimization problems such as the cutting stock problem [7] and the knapsack problem [10] can be modeled as BPPs [15]. Although there are many variants of this problem, in this investigation, we have focused on the one-dimensional online BPP (1D-BPP). Then, we assume that the only relevant property of the items is their length and that it is not possible to sort the items as a preprocessing step. An example of such a scenario is given by a production line with a fixed robot arm that packages items into the boxes. Here, items must be packed as they arrived, even if the whole production schedule can be known.

The current literature is vast in methods for solving the BPP [1,3,23]. Unfortunately, most of the methods that guarantee to find the optimal solution (also known as exact ones) are limited in the size of the instances they can handle. Conversely, approximation methods, such as heuristics, are fast to implement and execute but cannot guarantee the optimality of the solutions. More importantly, since these heuristics are usually generic methods, their performance may drastically change from one instance to the other, even within the same problem domain. A more robust way to tackle the BPP consists in combining the strengths of single heuristics, employing a hyper-heuristic (HH). A HH is a high-level method that decides when to use the individual heuristics throughout the solving process [4].

This idea of combining solvers dates back to the mid 70s [21]. From that moment onward, different solving strategies have emerged: algorithm portfolios [11], instance-specific algorithm configuration [17], and hyper-heuristics [20, 23], just to mention some. In general, these methods manage a set of solvers and apply the most suitable one for the problem instance. Aiming at unifying terms, from this point on, we will use the term "hyper-heuristic" to refer to the methods proposed in this paper.

In this work, we focus on developing score-based hyper-heuristics through a process that updates the scores of different heuristics according to their performance on a historical basis. Although a few studies have explored similar ideas in the past, the solution model proposed in this work is, to the best of the authors' knowledge, a novel approach. In the literature, we found no previous work that deals with the idea of training hyper-heuristics for the 1D-BPP by using such a straightforward reward-based strategy as the one described in this work.

We have organized the remainder of the document as follows. Section 2 presents the most relevant concepts and works related to this investigation. Section 3 details the hyper-heuristic model proposed in this work. In Section 4, we present the experiments conducted, their analysis, and discuss the most relevant findings. Finally, we present the conclusions and future work in Section 5.

2 Background

The 1-Dimensional Bin Packing Problem (1D-BPP) is defined by a set of n items and m bins, where w_j and c_j represent the length of item j and the capacity of each bin, respectively. To solve this BPP, it requires assigning each item to one bin such that the total weight of the items in each bin does not exceed c, and the number of bins is minimum.

The current literature contains significant examples of recent advances in solving the 1D-BPP. On the one hand, there are approaches based on metaheuristics. Abdel-Basset et al. [1] enhanced the Whale Optimization Algorithm (WOA) by adjusting positions inside the search space boundaries and implementing a

Lévy distribution to draw samples. Similarly, Zhang et al. [24] reported a buffered version of the next fit heuristic. The objective of the buffer is to store some items temporarily so that items with specific characteristics can be packed in the same bin. This allows controlling the wasted space of the bins on a similar range, and even filling the remaining open bins to its full capacity (whenever possible). In a more recent study, Gherboudj [18] adapted the African Buffalo Optimization (ABO) algorithm to solve the 1D-BPP. Their work combined four heuristics with the ABO algorithm to improve its behavior. This combination showed effective results in various test scenarios.

On the other hand, researchers have explored hyper-heuristics (HHs) to solve 1D-BPPs indirectly, because they work on the heuristic space rather than the solution one. When working with hyper-heuristics, they map the problem state through a set of features, so the most suitable heuristic can be applied. In the past, researchers have relied on metaheuristics, such as Genetic Algorithms (GAs) [19], Simulated Annealing (SA) [13], and Ant Colony Optimization (ACO) [6,9] to produce hyper-heuristics. Other authors have preferred machine learning techniques devoting considerable efforts to exploring supervised learning methods [5,16].

2.1 Heuristics

For this preliminary investigation, we have focused on two popular heuristics for solving the BPP: First Fit (FF) and Best Fit (BF). FF, as the name suggests, packs the next item in the first open bin, as long as it fits. One by one, all the bins are revised until it finds one where the item can be packed. If there is not such a bin, it opens a new one to pack the item there. Subsequently, BF looks for the bin with the minimum space to pack the item (*i.e.,* it minimizes the waste). As in FF, if no bin has enough space for the item, a new one is opened.

Forthwith, we describe how these heuristics work by using the instance depicted in Fig. 1. In this example, the next item to pack has a length of four units. Since two of those bins are full, they are considered to be closed. FF will try to pack this item as soon as possible. So, it packs it in the fourth bin since it has five units available. This action leads to a waste of one unit in such a bin. Conversely, BF will try to minimize the waste, looking for the bin where the item fits best. So, BF skips the fourth bin and packs the item into the fifth one, where it fits perfectly.

We are aware that there are many other popular heuristics available for solving the BPP, such as Djang and Finch and their multiple variants [22]. However, analyzing their effect went beyond the scope of this work.

2.2 Instances

In this work, we considered synthetic instances since they allow us to test the methods under specific and controlled scenarios. We now briefly describe them:

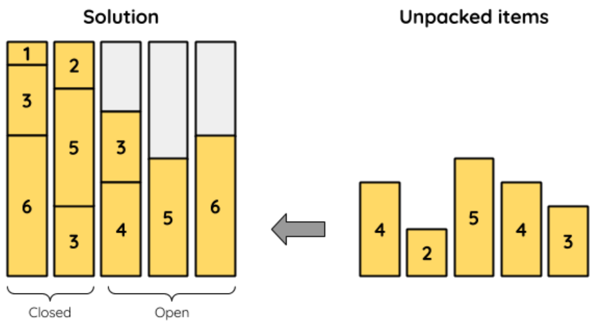

Fig. 1. Example of the solving process of a 1D-BPP instance of 15 items with lengths between one and six units and bins with a capacity of ten units. At the moment, ten items have been packed by using five bins (two closed and three open), and five items remain unpacked.

Training Set. It contains 100 small instances of 20 items with lengths between 1 and 32 units. The bins in these instances accept up to 64 units. This set has a mixture of instances so that no single heuristic performs the best in every instance. Such a fact forces the hyper-heuristic to switch between heuristics as the search takes place. It is noteworthy to mention that this situation might not hold for other sets of instances. To generate the Training Set, we used the evolutionary-based BPP generator introduced by Amaya et al. [2].

Test Set A. It consists of instances similar to the ones used for training. It contains 200 small instances of 20 items whose length varies between 1 and 32 units. The bin capacity is also defined at 64 units. To generate this set, we used the same generator from the previous experiment.

Test Set B. It incorporates the 160 instances proposed by Falkenauer [12]. These instances are classified into two different groups. The first one contains items with lengths uniformly distributed between 20 and 100 units and bins with a capacity of 150 units. The second group has items with sizes between 25 and 50 and bins of 100 units.

To characterize these instances, we used two dynamic features (they change throughout the solving process). These features are the proportion of open items concerning the total number of bins used (OBINS) and the average waste among all the open bins (AVGW). To exemplify how these features work, let us again take a look at the instance depicted in Fig. 1. Under these conditions, we calculate the value of OBINS as the number of open bins divided by the total number of bins. In other words, OBINS equals 3/5 for this example. Regarding the average waste, only open bins waste space. So, it sums 14 $(3 + 5 + 6)$ units. This way, the AVGW value for this instance is 14/5.

3 Model Description

Our hyper-heuristic (HH) model relies on a set of rules that works on two different levels: as a record the historical performance of single heuristics and as an estimation of their future performance. A rule has the purpose of finding a region on the problem space in which one specific heuristic behaves better than the others. Each rule contains two parts: a condition and an action. They are related to the instance state and the heuristic to apply, respectively.

Our HHs need to undergo a training process before we can use them in practical situations. During such a process, the HH iteratively updates the set of rules, one rule at a time. Then, when the hyper-heuristic deals with a new instance, the information within the rules determines which heuristic to use. The rules are different from the ones considered in other HH models, where the rule directly states the actions. In our approach, given a rule, we require an additional calculation to decide which heuristic to apply. The task of the training process is to find a set of rules that best discriminates the instance space. Fig. 2 depicts an example of how a rule looks inside the HH.

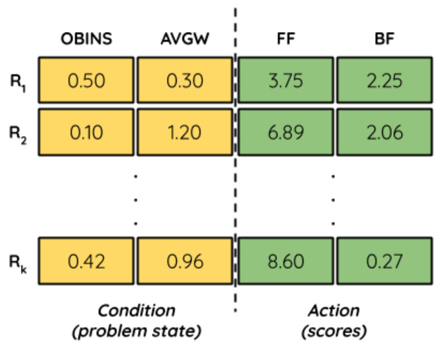

Fig. 2. An example of a hyper-heuristic produced by our solution model. Left: Condition contains the instance space description expressed in terms of the features OBINS and AVGW. Right: Action has the scores of the heuristics (*i.e.,* the larger, the better).

The model randomly initializes k rules by using a set of features that characterize the instance space as the condition and the available heuristics as the action of such rules. The scores of the heuristics in the rules are randomly initialized with values between 0 and 10. When the model deals with a new instance, it iteratively packs the items, one at a time. For each item, the hyper-heuristic decides which heuristic to apply. Let r be the rule with the condition closest to the instance state (via the Euclidean distance). Then, the heuristic with the largest score in the action of r is returned to pack the item. For example, given the rules depicted in Fig. 2 and an instance with values for OBINS and AVGW of 0.08 and 1.12, respectively, the rule with the closest condition to the current problem state is R_2. Now that we know that R_2 will be selected, the scores for FF and BF are 6.89 and 2.06, respectively. Based on these values, the rule will recommend using FF, since it has the highest score among all the heuristics in R_2.

It is important to remark that the aforementioned rules are updated *iff* the hyper-heuristic is on training mode. Such an update is performed in two different moments, as described:

– Every time the HH makes a decision, it is "rewarded" based on the quality of its decision. This process changes the scores of the selected rule (increases the scores related to the right decisions and decreases the ones related to the bad ones). To decide the value to add or subtract to the scores, we calculate the reward as $0.01/(\text{OBINS} \times \text{AVGW})$.
– After the HH has packed all the items in the instance, the system replaces the less used rule. The system generates a new rule according to one out of three initialization functions: RANDOM (a random choice for the conditions), MEAN (the average values of the points visited during the search of the last instance), and LAST (the values of the features of the last visited point in the instance space). In all cases, the scores for the heuristics are randomly initialized by using a uniform distribution function between 0 and 10.

4 Experiments and Results

In this investigation, we conducted two experiments. The first one explored the generation of hyper-heuristics by using three strategies to initialize the rules. For each strategy, we generated 11 hyper-heuristics by using the approach described in Section 3. Subsequently, the second experiment compared the performance of the best hyper-heuristics produced from the first experiment on a set of instances with features different from the ones used for training. In other words, the second experiment analyzed the behavior of the most competent hyper-heuristics produced by our solution model on a more realistic and challenging scenario.

4.1 Exploratory Experiment

The objective of this experiment was to evaluate three different methods for initializing rules and their decisions. In all cases, hyper-heuristics contained 20 rules. This value was decided based on our previous experience with this problem. The training process to produce one hyper-heuristic ran for 100 epochs in each case —an epoch occurs when all the instances in the Training Set have been analyzed. We obtained 11 hyper-heuristics by using each initialization function (*i.e.,* RANDOM, MEAN, and LAST). Then, the two heuristics and the 33 hyper-heuristics were applied on the Test Set A. In all cases, the average waste across all the items (AVGW) was used to estimate the quality of the solutions.

Fig. 3 presents the resulting data. Among the hyper-heuristics produced, there are two worth analyzing in more detail. Let us call these hyper-heuristics

HHA and HHB. The former represents the hyper-heuristic that performed best when initializing them with the MEAN approach. Conversely, HHB represents the best performing hyper-heuristic. It is worth remarking that HHB corresponds to the LAST initialization. While FF and BF produced an average waste of 8.32 and 5.07 units in Test Set A, HHA and HHB reduced the average waste to 4.62 and 3.87 units, respectively. Despite these savings, both of them are still far from the Oracle, which produced an average waste of only 2.76 units for the same set.

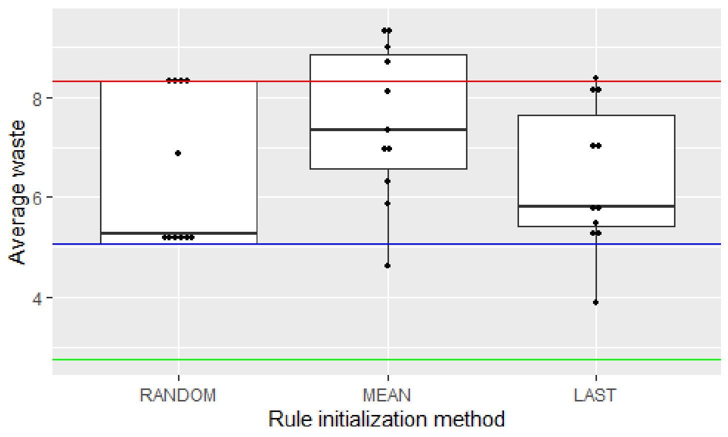

Fig. 3. Average waste on the Test Set A produced by 11 hyper-heuristics generated with each rule initialization method. The red, blue, and green horizontal lines represent the average waste of First Fit (FF), Best Fit (BF), and the Oracle (the best performer for each instance), respectively.

We now consider the proportion of instances where the solving strategies behave as competent as the Oracle. Let us refer to such a proportion as the success rate. Then, the better the solving strategy, the closer to 100% the success rate becomes. The reason: this would mean that the strategy performed as the Oracle on every instance within the set. The success rate of FF and BF on the Test Set A was 50% and 77.5%, respectively. Conversely, the success rate of HHA and HHB on the same set increased to 82% and 89.5%, respectively.

4.2 Confirmatory Experiment

We are now interested in observing the behavior of the best hyper-heuristics, HHA and HHB, on instances whose features differ from those of the ones used for training. So, we use them to solve the Test Set B. Such a set contains the instances generated by Falkenauer [12], as mentioned before. Once again, performance data are compared against that of the Oracle. Based on the results obtained, we observed many instances where both FF and BF behave in the same way (88.13% of the instances). Nonetheless, both hyper-heuristics (HHA and HHB) proved to be reliable and competent. HHA obtained a success rate of 95%, while HHB obtained 97.5%. These values mean that these hyper-heuristics were unable to replicate the Oracle in only 8 and 4 out of the 160 instances, respectively.

4.3 Discussion

The analysis conducted on three different initialization methods (*i.e.,* RANDOM, MEAN, and LAST) suggests that a simple random function is not powerful enough to outperform individual heuristics. On a closer look, we found that RANDOM tends to replicate the behavior of the single heuristics. Thus, it seems that it reduces the capability of the hyper-heuristic to discriminate between heuristics and alternate their use throughout the solving process. MEAN and LAST methods demonstrate that it is indeed possible to improve the results of the heuristics throughout the proposed approach. Among the three initialization methods, on average, LAST performed best. However, better techniques can likely be found by deepening the study on this matter.

When testing the two best HHs on the Falkenauer dataset [12], we observed that these hyper-heuristics remain competitive, although they were not trained for such instances. The two hyper-heuristics that we analyzed in more detail (HHA and HHB, produced with MEAN and LAST, respectively) were close to fully replicating the behavior of the Oracle, proving its contribution. The difference between Oracle and these hyper-heuristics lies in the way decisions are made. While the Oracle solves an instance with the same heuristic from start to end, HHA and HHB use different heuristics depending on their decision rules. This result means that there may be many different paths that lead to similar and high-quality solutions.

5 Conclusion and Future Work

Throughout this study, we proposed a score-based hyper-heuristic (HH) model for tackling the 1-Dimensional Bin Packing Problem (1D-BPP). This model iteratively updates its internal structure to capture the patterns in the instance space, which suggests when one heuristic performs better than the others. Therefore, the proposed model generates a set of rules that segments the 1D-BPP instance space. This segmentation allows the system to discriminate between heuristics throughout the search, as a means to improve the quality of the solutions. Our findings suggest that the initialization method is crucial for this model to work. By merely using arbitrary rules, the model does not reach a competitive solution against a single heuristic. Moreover, from the three methods we tested, LAST behaved best.

It is imperative to remark that this document describes the first study towards a score-based HHs —at least in the way we propose it. Unfortunately, due to space restrictions, we could only consider two heuristics and two features to characterize the instance space. Moreover, for this preliminary investigation, we wanted to keep a basic set of heuristics to estimate the contribution of the proposed approach. Naturally, we expect to extend this idea and cover more heuristics as part of future work. When new heuristics and features are introduced, the model might behave differently. However, it is noticeable that our proposed model requires no changes to incorporate more heuristics or features, so it can quickly scale to more complex situations.

As part of the future work, we would like to explore how this model behaves on a different problem domain, such as the knapsack problem or the graph coloring problem, which are some exciting and challenging combinatorial optimization problems we would like to address through hyper-heuristics. Also, all the instances considered for this work were synthetic, but other kinds are required to validate the contributions of our approach thoroughly. We also plan on incorporating them as a future step of this work.

Acknowledgments

This research was partially supported by CONACyT Basic Science Project under grant 287479 and ITESM Research Group with Strategic Focus on Intelligent Systems.

References

1. Abdel-Basset, M., Manogaran, G., Abdel-Fatah, L., Mirjalili, S.: 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. Amaya, I., Ortiz-Bayliss, J.C., Conant-Pablos, S.E., Terashima-Marín, H., Coello Coello, C.A.: Tailoring instances of the 1d bin packing problem for assessing strengths and weaknesses of its solvers. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.) Parallel Problem Solving from Nature – PPSN XV. pp. 373–384. Springer, Cham (2018)
3. Asta, S., Özcan, E., Parkes, A.J.: CHAMP: Creating heuristics via many parameters for online bin packing. Expert Systems with Applications **63**, 208–221 (Nov 2016)
4. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society **64**, 1695–1724 (2013)
5. Choong, S.S., Wong, L.P., Lim, C.P.: Automatic design of hyper-heuristic based on reinforcement learning. Information Sciences **436**, 89–107 (2018)
6. Cuesta-Cañada, A., Garrido, L., Terashima-Marín, H.: Building hyper-heuristics through ant colony optimization for the 2d bin packing problem. In: Khosla, R., Howlett, R.J., Jain, L.C. (eds.) Knowledge-Based Intelligent Information and Engineering Systems. pp. 654–660. Springer, Berlin, Heidelberg (2005)
7. Delorme, M., Iori, M., Martello, S.: Bin packing and cutting stock problems: Mathematical models and exact algorithms (Nov 2016)

8. 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)
9. Duhart, B., Camarena, F., Ortiz-Bayliss, J.C., Amaya, I., Terashima-Marín, H.: An experimental study on ant colony optimization hyper-heuristics for solving the knapsack problem. In: Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Olvera-López, J.A., Sarkar, S. (eds.) Pattern Recognition. pp. 62–71. Springer (2018)
10. Eliiyi, U., Eliiyi, D.T.: Applications of bin packing models through the supply chain. International journal of business and management **1**(1), 11–19 (Jun 2009)
11. Epstein, S.L., Freuder, E.C., Wallace, R., Morozov, A., Samuels, B.: The adaptive constraint engine. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming. pp. 525–542. CP'02, Springer, London, UK (2002)
12. Falkenauer, E.: A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics **2**(1), 5–30 (1996)
13. Garza-Santisteban, F., Sánchez-Pámanes, R., Puente-Rodríguez, L.A., Amaya, I., Ortiz-Bayliss, J.C., Conant-Pablos, S., Terashima-Marín, H.: A simulated annealing hyper-heuristic for job shop scheduling problems. In: 2019 IEEE Congress on Evolutionary Computation (CEC). pp. 57–64 (Jun 2019)
14. Hu, H., Zhang, X., Yan, X., Wang, L., Xu, Y.: Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method. arXiv preprint (Aug 2017)
15. 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)
16. Lara-Cárdenas, E., Sánchez-Díaz, X., Amaya, I., Ortiz-Bayliss, J.C.: Improving hyper-heuristic performance for job shop scheduling problems using neural networks. In: Mexican International Conference on Artificial Intelligence. pp. 150–161. Springer (2019)
17. Malitsky, Y.: Evolving instance-specific algorithm configuration. In: Instance-Specific Algorithm Configuration, pp. 93–105. Springer (2014)
18. Odili, J.B., Kahar, M.N.M., Anwar, S.: African Buffalo Optimization: A Swarm-Intelligence Technique. In: Procedia Computer Science. vol. 76, pp. 443–448. Elsevier B.V. (Jan 2015)
19. Ozcan, S.O., Dokeroglu, T., Cosar, A., Yazici, A.: A novel grouping genetic algorithm for the one-dimensional bin packing problem on GPU. In: Communications in Computer and Information Science. vol. 659, pp. 52–60. Springer (Oct 2016)
20. Pillay, N., Qu, R.: Hyper-Heuristics: Theory and Applications. Natural Computing Series, Springer (2018)
21. Rice, J.R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976)
22. Sim, K., Hart, E., Paechter, B.: A Hyper-Heuristic Classifier for One Dimensional Bin Packing Problems: Improving Classification Accuracy by Attribute Evolution. In: Lecture Notes in Computer Science, pp. 348–357. Springer (2012)
23. Sim, K., Hart, E., Paechter, B.: A Lifelong Learning Hyper-heuristic Method for Bin Packing. Evolutionary Computation **23**(1), 37–67 (Mar 2015)
24. Zhang, M., Lan, Y., Li, H.: A New Bin Packing Algorithm with Buffer. In: 2018 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS). pp. 625–628. IEEE (Jan 2018)