symmetry

Article

Identify Patterns in Online Bin Packing Problem: An Adaptive Pattern-Based Algorithm

Bingchen Lin [1], Jiawei Li [1,*], Ruibin Bai [1], Rong Qu [2], Tianxiang Cui [1] and Huan Jin [1]

1 School of Computer Science, University of Nottingham Ningbo China, Ningbo 315000, China; bingchen.lin@nottingham.edu.cn (B.L.); ruibin.bai@nottingham.edu.cn (R.B.); tianxiang.cui@nottingham.edu.cn (T.C.); huan.jin@nottingham.edu.cn (H.J.)
2 School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK; rong.qu@nottingham.ac.uk
* Correspondence: jiawei.li@nottingham.edu.cn

Abstract: Bin packing is a typical optimization problem with many real-world application scenarios. In the online bin packing problem, a sequence of items is revealed one at a time, and each item must be packed into a bin immediately after its arrival. Inspired by duality in optimization, we proposed pattern-based adaptive heuristics for the online bin packing problem. The idea is to predict the distribution of items based on packed items, and to apply this information in packing future arrival items in order to handle uncertainty in online bin packing. A pattern in bin packing is a combination of items that can be packed into a single bin. Patterns selected according to past items are adopted and periodically updated in scheduling future items in the algorithm. Symmetry in patterns and the stability of patterns in the online bin packing problem are discussed. We have implemented the algorithm and compared it with the Best-Fit in a series of experiments with various distribution of items to show its effectiveness.

Keywords: bin packing problem; heuristic; pattern recognition; stability

check for updates

Citation: Lin, B.; Li, J.; Bai, R.; Qu, R.; Cui, T.; Jin, H. Identify Patterns in Online Bin Packing Problem: An Adaptive Pattern-Based Algorithm. *Symmetry* **2022**, *14*, 1301. https://doi.org/10.3390/sym14071301

Academic Editor: Aviv Gibali

Received: 24 May 2022
Accepted: 19 June 2022
Published: 23 June 2022

1. Introduction

Combinatorial optimisation problems have extensive applications in many real-world applications. However, most of them are NP-hard and finding the optimal solutions becomes computationally prohibitive because of combinatorial explosion. The existing approaches to these problems can broadly be classified into analytical-model-driven methods and data-driven methods. Analytical-model-driven methods focus on the analytical properties of the mathematical model but suffer from robustness issues over uncertainties from the input data. Heuristic algorithms generally compute feasible solutions with low computational complexity, not optimum. Data-driven methods often formulate the combinatorial problems as online optimization problems and try to tackle the problem in a generative fashion. One of the main drawbacks of the these data-driven methods is their inability to efficiently exploit the core structures and properties of the problem. More specifically, existing data-driven methods primarily focus on the objectives to be optimized, but neglect various complex inter-dependencies among the decision variables in the form of constraints. This can be illustrated with the following example for the bin packing problem.

A one-dimensional bin packing problem aims to use the minimum number of bins of identical size to pack a set of items of different sizes. In its offline version, the sizes of the items are given prior to the packing. Let B denote the capacity of the bins to be used. The problem is to pack all the items of n types, with each item type i having a size s_i and quantity q_i. Let y_j be a binary variable to indicate whether bin j is used in a solution ($y_j = 1$) or not ($y_j = 0$). Let x_{ij} be the number of times item type i is packed in bin j. The problem can be formulated as follows:

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{j=1}^{U} y_j && && (1) \\ \text{subject to:} \quad & \textstyle\sum_{j=1}^{U} x_{ij} = q_i && \text{for } i = 1, \cdots, n && (2) \\ & \textstyle\sum_{i=1}^{n} s_i x_{ij} \leq B y_j && \text{for } j = 1, \cdots, U && (3) \end{aligned}$$

where U is the maximal number of possible bins available to use.

In an online bin packing problem, we consider an infinite sequence of items without prior knowledge of their sizes. Each item needs to be packed at the time of arrival. Heuristic algorithms are commonly used to solve this kind of problem with uncertainty, since optimal solutions are impossible to acquire.

If the distribution of items in an online BPP is known, it is possible to acquire near-optimal solutions by utilizing such knowledge. A straightforward idea is to compute the distribution of arrival items via statistics and then apply the distribution in packing forthcoming items. We have developed a pattern-based heuristic algorithm for BPP based on the assumption that patterns maintain stability corresponding to distribution of items. A pattern denotes a combination of items fitting for packing in a bin. Consider the example in Table 1.

Table 1. An example of online BPP. s_i is item type size and $B = 10$.

BPP	$\{s_i\} = \{2,4,5,3,2,2,2,3,3,4,2,4,5,3,2,2,2,3,3,4\}$
Solutions	Best-Fit: $\{2,4,3\},\{5,2,2\},\{2,3,3,2\},\{4,4,2\},$ $\{5,3,2\},\{2,3,3\},\{4\}$ opt. solution: $\{2,5,3\},\{4,2,4\},\{2,2,3,3\}$ $\{2,5,3\},\{4,2,4\},\{2,2,3,3\}$

The well-known Best-Fit heuristic produces sub-optimal solution for the case. It can be seen that the optimal solution contains patterns $\{2,5,3\}$, $\{4,2,4\}$, and $\{2,2,3,3\}$. If an algorithm learns the patterns from statistics of the packed items and applies the information in packing the remaining items, it will be able to implement solutions close to the optimal solution if the following sequence of items satisfies the same distribution.

In this paper, we propose an adaptive pattern-learning algorithm for online BPP that identifies patterns, dynamically updates the item size distribution, and plans the future packing in an adaptive manner. The contribution of this study is threefold. First, it is the first time that the distribution of items is predicted and used in packing via patterns. The idea of retrieving information from packed items and using the information in planning future items may be applied to other optimization problems with uncertainty. Second, we analyze the stability of patterns by showing that a small error in the estimation of item distribution will only cause small deviation from the optimal solution. Third, an algorithm that identifies and updates patterns is developed for online BPP, which outperforms Best-fit on average bin usages and computational complexity for a large set of benchmark instances.

2. Literature Review

Bin-Packing Algorithms

In this section, a detailed definition of the online bin-packing problem would be summarized as a foundation, and competitive ratio, a widely used measurement for algorithm performance, would be introduced. Reviews of several classical algorithms would also be given.

The classical bin packing problem is defined by an infinite supply of bins with capacity B and a list L of k items (or elements). Recall that the aim is to pack the items into a minimum number of bins under the constraint that the sum of the sizes of the items in each bin is no greater than B. A bin-packing problem is called "online" when each element needs to be packed as soon as it is inspected, without the knowledge of other future items.

The performance of an approximation algorithm is majorly measured by its worst case behavior, which could be quantified by an asymptotic worst-case ratio (asymptotic performance ratio). For online problems such as the online bin-packing problem, it is also called a competitive ratio, which would be used in this paper. Let $A(L)$ denotes the number of bins used by Algorithm A to pack the items of L; OPT denotes an optimal algorithm, which always uses a minimal number of bins. Let V_α be the set of all lists L for which the maximum size of the items is bounded from above by α. For every $k \geq 1$,

$$R_A(k,\alpha) = \sup_{L \in V_\alpha} \left\{ \frac{A(L)}{k} : OPT(L) = k \right\} \tag{4}$$

Based on Equation (4), the competitive ratio (or asymptotic performance ratio) could be formulated as a function of α:

$$R_A^\infty(\alpha) = \overline{\lim}_{k \to \infty} R_A(k,\alpha) \tag{5}$$

As shown by Equation (5), $R_A^\infty(\alpha)$ measures the quality of the packing decisions made by algorithm A in the worst case, compared with the optimal result. Additionally, plainly, $R_A^\infty(\alpha) \geq 1$. Based on Equations (4) and (5), a more universal form could be derived, which is

$$A(L) \leq R_A^\infty(\alpha) OPT(L) + K \tag{6}$$

for every list $L \in V_\alpha$. Conventionally, if α is not specified, the competitive ratio of algorithm A could be denoted as $R_A^\infty \equiv R_A^\infty(1)$. In this paper, uncommon cases where $\alpha \neq 1$ are not studied, so no specification would be made.

Based on the measure above, many simple but effective online algorithms were proposed. In describing an online bin-packing algorithm, we use the current item to indicate the next item to be packed before a decision point. So right after a packing decision, the item current item changes from a_i to a_{i+1}. One of the simplest approaches is to pack items in sequence according to

Next-Fit (NF): After packing the first item, NF packs each successive item in the last opened bin, if the bin could contain that item. Otherwise, the last opened bin would be closed and the current item would be placed in a new empty bin.

The advantage of Next-Fit is that it is quite an efficient algorithm, with the time complexity of O(n). The disadvantage of this algorithm is a relatively poor competitive ratio: $R_{NF}^\infty = 2$ [1].

A conspicuous aspect of its performance is that it closes bins that could be used for packing future items. A direct modification to this algorithm is to keep all the bins open throughout the packing process. A greedy way is to locate a bin with largest remaining capacity, which is

Worst-Fit (WF): If there is no open bin that can contain the current item, then WF packs the item into a new empty bin. Otherwise, WF packs the current item into a bin with the largest remaining capacity. If there is more than one such bin, WF chooses the one with the lowest index.

Although it could be expected that WF performs better than NF, it does not. It is proved that $R_{WF}^\infty = R_{NF}^\infty$ [2].

Another simple rule is to scan through all the opened bins one by one and put the item into the first bin that fits the item.

First-Fit (FF): If there is no open bin that could contain current item, then FF packs the item into a new empty bin. Otherwise, FF packs the current item into the lowest-indexed bin that fits.

To achieve a better performance, a natural complement to WF has also been studied. Instead of choosing the bin with highest remaining capacity, it aims to pack the item into a bin such that the waste is minimized.

Best-Fit (BF): If there is no open bin that could contain current item, then WF packs the item into a new empty bin. Otherwise, BF packs the current item into a bin with the lowest remaining capacity. If there is more than one such bin, BF chooses the one with the lowest index.

By adopting proper data structure, the time complexity of FF, BF and WF is O(n log n). It has been proved that $R^{\infty}_{FF} = R^{\infty}_{BF} = 1.7$ [1].

FF, BF, and WF share many common characteristics. One of them is the satisfaction of Any-Fit constraint, proposed by [ref]. Any-Fit constraint is that, in a packing decision, if bin j is empty, it cannot be chosen unless the current item will not fit in any bin to the left of bin j. By fulfilling the constraints, FF, BF, and WF are also called Any-Fit algorithms. They are straightforward, incremental, and do not classify open bins into different categories. The upper and lower bound of competitive ratio of these algorithms have been proved, which are $R^{\infty}_{FF} \leq R^{\infty}_{A} \leq R^{\infty}_{WF}$ [2].

Other than incremental Any-Fit algorithms, many bounded-space algorithms have also been explored. An algorithm is bounded-space if the number of open bins at any time in the packing process is bounded by a constant. For example, First-Fit utilizes one bounded-space. The motivation and practicality of developing this type of algorithm are clear. For example, to load trucks at a depot, one cannot have an infinite number of trucks at the loading dock. One of the most trivial algorithms is a modified version of Next-Fit called Next-k-Fit ($\mathrm{NF_k}$) [3]. It packs items following the First-Fit rule, but only considers k most recently opened bins; when a new bin has to be opened, the opened bin with lowest index would be closed. As expected, the competitive ratio of $\mathrm{NF_k}$ tends to 1.7 with k increasing. Based on the idea of bounded-space, a potential improvement direction, as pointed out by [4] and summarized by [5], is to consider adopting the reservation technique, to proactively retain empty space for future items. This idea had already appeared in some research.

Ref. [6] proposed a bounded-space algorithm $\mathrm{Harmonic_k}(H_k)$. This algorithm is based on a special, non-uniform partitioning of item size interval $(0, 1]$. The "harmonic partitioning" is used to classify items into k groups: $I_j = \left(\frac{1}{j+1}, \frac{1}{j}\right](1 \leq j \leq k-1)$ and $I_k = \left(0, \frac{1}{k}\right]$. I_j-element is defined as an element of which the size belongs to interval I_j. Similarly, bins are also classified into k categories and an I_j-bin is defined to contain I_j-element only. I_j-elements could only be packed into an I_j-bin following the rule of Next-Fit, and thus, at most k bins could be open at the same time. It has been proved that $R^{\infty}_{H_\infty} \approx 1.69103$, lower than all Any-Fit algorithms. The bottleneck of the algorithm is the waste of space of I_1-element, of which the range of size is $\left(\frac{1}{2}, 1\right]$. For an item with size just over $\frac{1}{2}$, almost half of the capacity of the packed bin would be wasted. However, it has been proved that bounded-space algorithm cannot do better than $\mathrm{Harmonic_k}$.

Yao [4] firstly broke through this barrier with the Refined First-Fit (RFF) algorithm, which has unbounded space. Unlike $\mathrm{Harmonic_k}$, this algorithm does not parameterize the partitioning but divides the $(0, 1]$ statically into $\left(0, \frac{1}{3}\right]$, $\left(\frac{1}{3}, \frac{2}{5}\right]$,$\left(\frac{2}{5}, \frac{1}{2}\right]$,and$\left(\frac{1}{2}, 1\right]$. Other than packing type$-i$ item into type$-i$ bin, every sixth type-2 item would be packed by First-Fit into type-4 bins. It has been proved that $R^{\infty}_{RFF} = \frac{5}{3} \approx 1.666$. Compared with $\mathrm{Harmonic_3}$, RFF separates the $\left(\frac{1}{3}, \frac{1}{2}\right]$with $\frac{2}{5}$, and assigns $\left(\frac{1}{3}, \frac{2}{5}\right]$ to complement the bin space wasted by $\left(\frac{1}{2}, 1\right]$ items.

With the idea from RFF, many algorithms based on $\mathrm{Harmonic_k}$ managed to better use the wasted space of I_1-bins, including Refined Harmonic [6], Modified Harmonic [7] Harmonic+1 [8], Harmonic++ [9], etc. The latest lower bound of this problem is 1.54037 [10].

Like H_k and RFF, every algorithm with the idea of a reservation technique tends to combine two or more types of items with some rules to avoid being short sighted. H_k combines every I_j-element into I_j-bins so that all the closed bins must have a remaining capacity less than $\frac{1}{j+1}$. For example, a closed I_5-bins with item size range $\left(\frac{1}{6}, \frac{1}{5}\right]$ always have five items inside, which means that the utilized capacity would be higher than $\frac{5}{6}$, which means that the wasted space would always be less than $\frac{1}{6}$. It is clear that as j

increasing, the utilisation of bins would increase. The calculation of competitive ratio of this algorithm [6] shows this trend: as k rises, the competitive ratio never worsens.

Recall that based on the limitation of the wasted space caused by large items, especially ones larger than $\frac{1}{2}$, one improvement is to fill the wasted space with other items. Very recently, after many Harmonic-based algorithms, Ref. [11] summarized a general framework SuperHarmonic: it classifies incoming items of one type into red items and blue items. Then, intuitively, it packs the blues bottom up and packs the reds top down to reduce waste. Other than harmonic partitioning, it also integrates manual partitioning. Based on this framework, algorithm SonOfHarmonic reaches the competitive ratio of 1.5813.

Other than harmonic-based approaches, many methods with various performance measures have been developed for BPP. Attempting to combine strength of different heuristics, hyper-heuristics that chose one suitable heuristic from a set of heuristics were proposed to solve a particular portion of a problem instance [12]. The performance of different heuristics on a historical basis was measured and a decision was made on heuristics selection for the next segment of instance. The success rate, compared to BF and FF, was adopted as a performance measure.

An adaptive rule-based algorithm was developed in [13]. Instead of using a value to choose different heuristics, it automatically configured the behavior of an algorithm by computing a threshold value based on available data elements. This could be seen as a generic meta-heuristic framework and could be applied to multiple online problems, including bin packing, lot sizing and scheduling. The algorithm was assessed using an experimental competitive ratio.

Inspired by approximate interior-point algorithms for convex optimization, Ref. [14] proposed a interior-point-based algorithm. It is fully distribution-oblivious, meaning that it does not make decisions based on distribution information, i.e., it is learning based. Under its average case analysis, the algorithm was proven to exhibit $\mathcal{O}(\sqrt{T})$ additive suboptimality compared to offline solutions, which is lower than all existing distribution-oblivious algorithms, which have $\mathcal{O}(T)$.

In [15] attempts were made to directly apply human intuition to solve 2D BPP. They recorded human behaviors from game-play and created multiple Human-Derived Heuristics (HDH) in the form of decision trees.

A Deep Reinforcement Learning (DRL) method was adopted learn the policies to solve a 3D online BPP [16]. Since the target scenario is the factory assembly line, the next few coming items could be determined in advance with this problem formulation. Space utilization and stacking stability were used as performance measures.

Little research has been conducted to establish a full understanding of combining a parameterized partition set of items to form patterns. This will be investigated in the following sections.

3. Patterns in Bin Packing Problem

The concept of patterns is used in various domains of computer science: design patterns in software engineering, traditional pattern recognition in signal processing, etc. To some extent, they aim to summarize an abstract combination or paradigm from various instances or specific examples, and to be reused for advantageous performance or efficiency in computation or engineering. In BPP, patterns could also be applied to accelerate the packing process and improve the space utilization of bins. Consider a simple example below (Table 2).

Table 2. A simple example of online BPP. s_i is item type size and $B = 1$.

BPP	$\{s_i\} = \{0.33, 0.32, 0.41, 0.49, 0.59, 0.58\}$
Solutions	Best-Fit: $\{0.33, 0.32\}, \{0.41, 0.49\}, \{0.59\}, \{0.58\}$
	Harmonic$_3$: $\{0.33, 0.32\}, \{0.41, 0.49\}, \{0.59\}, \{0.58\}$
	opt. solution: $\{0.33, 0.58\}, \{0.41, 0.49\}, \{0.32, 0.59\}$

In this example, Best-Fit and Harmonic$_3$ use four bins. The optimal solution uses three bins. As the first bin in the solution of Harmonic$_3$ has reserved space for other I_1 items, it is not fair in such a small problem instance. However, based on direct inspection of the item distribution, it could be found that there are two items in $(0.3, 0.4]$ and two items in $(0.5, 0.6]$. If we have performed some bin packing with a similar item size distribution, we could derive this knowledge from history item size distribution and apply it in packing, and an optimal solution may be easier to find.

The reservation technique adopted by several Harmonic-based algorithms also implies an implicit adoption of patterns. However, their designs are based on asymmetrical harmonic partitioning rather than a fully symmetrical design. They also do not have a mechanism to apply prior knowledge. Our work would be based on a symmetrical design of partitions and distribution-based patterns.

3.1. Discretize Items

In our algorithm, similar to the Harmonic-based algorithm, an item is given a type based on its size. In Harmonic-based algorithms, the partitioning is either purely based on harmonic partitioning or a mixture of manual partitioning and harmonic partitioning. We use symmetrical partitioning in this study. We divide $(0, 1]$ into N sections, where N is a whole number. We use S_i to represent the i-th section, $i = 1, 2, \ldots, N$, the range of which is $(\frac{i-1}{N}, \frac{i}{N}]$. A direct comparison is give in Table 3.

Table 3. Partitioning comparison: H_k vs. our algorithm.

i	I_i in H_7	S_i in Our Algorithm with $N = 7$
1	$(\frac{1}{2}, 1]$	$(0, \frac{1}{7}]$
2	$(\frac{1}{3}, \frac{1}{2}]$	$(\frac{1}{7}, \frac{2}{7}]$
3	$(\frac{1}{4}, \frac{1}{3}]$	$(\frac{2}{7}, \frac{3}{7}]$
4	$(\frac{1}{5}, \frac{1}{4}]$	$(\frac{3}{7}, \frac{4}{7}]$
5	$(\frac{1}{6}, \frac{1}{5}]$	$(\frac{4}{7}, \frac{5}{7}]$
6	$(\frac{1}{7}, \frac{1}{6}]$	$(\frac{5}{7}, \frac{6}{7}]$
7	$(0, \frac{1}{7}]$	$(\frac{6}{7}, 1]$

A pattern is defined as a collection of types of items to be packed into a bin. In classical H_k algorithm, types of items are packed in a bin, which means that, for I_i items, the pattern is $i - 1$ items in a bin. In our algorithm, various types of items are combined as a pattern. Patterns are listed in a large-item-first manner. P_i stands for the i-th pattern. For example, when $N = 17$, the first nine patterns are listed in Table 4.

Table 4. Example patterns with 17 sections.

i	P_i
1	$\{S_{17}\}$
2	$\{S_1, S_{16}\}$
3	$\{S_2, S_{15}\}$
4	$\{S_1, S_1, S_{15}\}$
5	$\{S_3, S_{14}\}$
6	$\{S_1, S_2, S_{14}\}$
7	$\{S_1, S_1, S_1, S_{14}\}$
8	$\{S_4, S_{13}\}$
9	$\{S_1, S_3, S_{13}\}$

Compared with Harmonic, our method of partitioning and constructing patterns could adapt the incoming items distribution, of which the detailed algorithm would be explained in Section 4.

The quality of a pattern could be measured by the wasted space of a bin after packing. It could help determine the priority of patterns and other parameter settings for the packing process. It could be worst-case wasted space based solely on the pattern itself, or average-case worsted space relying on both the pattern and the distribution. To compute the average case, knowledge of the item size distribution is required. Algorithm A2, which would be explained in Section 4, could be used to compute the average-case wasted space. As the item size distribution is usually unknown before the whole packing process started, worst-case wasted space is more applicable for analysis.

For worst-case wasted space, we could easily find the maximum space wasted by a bin packed with the pattern. If we divide $(0, 1]$ into N sections, then the range of a section is of width $\frac{1}{N}$. If we have a pattern with n items included, then the range of occupied space of a bin packed by the pattern would be of width $n \times \frac{1}{N}$. The worst-case wasted space is $1 - n \times \frac{1}{N}$. It could be derived that, as n increases, more space would be wasted; as the N increases, less space would be wasted. This implies that, to achieve higher performance, the number of sections needs to be set sufficiently high; and patterns containing less items could be given higher priority for matching.

3.2. Symmetry in Patterns

As the sizes of items are evenly categorized, we find symmetry in distribution of patterns in solutions. We encode patterns as integer arrays with number of sections. The i-th integer in an array is the number of items in section i in the corresponding pattern. In this way, we could construct an extended table to include patterns' array representations (Table 5). Note that this array representation of pattern would also be used in the algorithm.

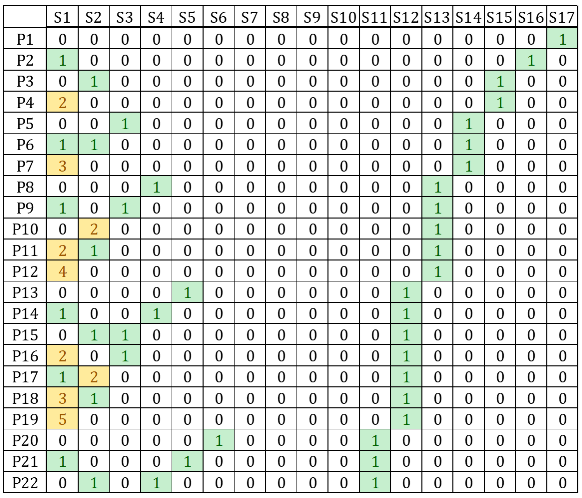
Table 5. Example patterns with 17 sections and array representation.

i	P_i	P_i **Array Representation**
1	$\{S_{17}\}$	$[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]$
2	$\{S_1, S_{16}\}$	$[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]$
3	$\{S_2, S_{15}\}$	$[0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]$
4	$\{S_1, S_1, S_{15}\}$	$[2,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]$
5	$\{S_3, S_{14}\}$	$[0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0]$
6	$\{S_1, S_2, S_{14}\}$	$[1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]$
7	$\{S_1, S_1, S_1, S_{14}\}$	$[3,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]$
8	$\{S_4, S_{13}\}$	$[0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0]$
9	$\{S_1, S_3, S_{13}\}$	$[1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0]$

Based on this representation, a table (Figure 1) could be constructed by stacking rows of patterns. The height of this table is the number of patterns, whereas the width of it is the number of sections. To intuitively reveal the symmetry, colors are added to some cells in the table: green for one item in the section and yellow for more than one item in the section.

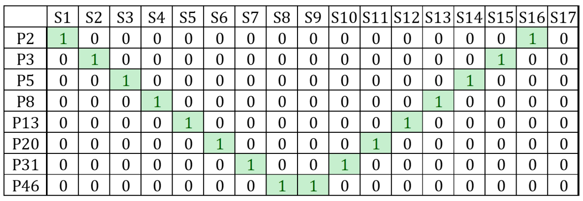
The patterns with two items are prioritized in pattern selection, since these patterns do not contain common items (Figure 2). Any change in the number of one pattern does not have an influence on other patterns. A simple vertical symmetry could be found in the chart, where the axis of symmetry lies between S_8 and S_9.

Alternatively, we have defined sub-prioritized patterns that contain three items in each pattern (Figure 3). As marked by bold borders, a similar pattern of vertical symmetry appears in each rectangle.



	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17
P1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
P2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
P3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P5	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
P6	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
P7	3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
P8	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
P9	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P10	0	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P11	2	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P12	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P13	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
P14	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
P15	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P16	2	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P17	1	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P18	3	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P19	5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P20	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
P21	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
P22	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0

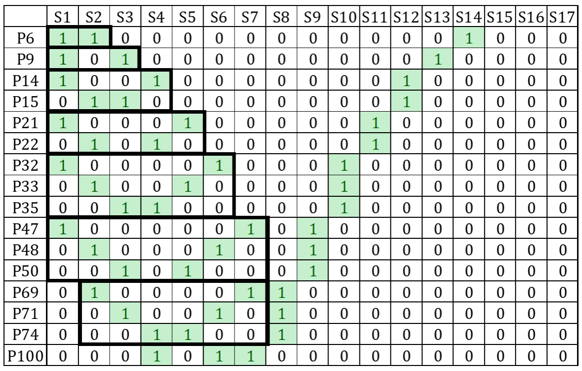
Figure 1. Colored original pattern distribution. (First 22 patterns; green cells indicate one item in the section; yellow cells indicate more than one item in the section).



	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17
P2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
P3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P5	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
P8	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
P13	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
P20	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
P31	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
P46	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Figure 2. Prioritized patterns (green cells).

It could be hypothesized that symmetry appears regularly in the pattern discretion. As the number of sections increases, there are symmetrical patterns in different sets of pattern distribution. It is non-trivial to understand this feature: when the maximum number of items of a section of a pattern is restricted to 1, after we set the section count N, all patterns have a constraint of total size in sections of N. If we have M items including item A and item B, and sizes of other $(M-2)$ items are determined, then $A+B$ would be a constant under this condition. As A becomes bigger, B would become smaller.



	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17
P6	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
P9	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P14	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
P15	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P21	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
P22	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
P32	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
P33	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
P35	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
P47	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
P48	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
P50	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0
P69	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
P71	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
P74	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0
P100	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0

Figure 3. Sub-prioritized patterns (The green cells in black bold boxes).

This feature gives our algorithm an advantage in handling symmetrical distributions. For example, Figure 4. shows a normal item size distribution. If we set 10 sections for partitioning, it could be seen that the frequency in S_{5-i} and S_{5+i} is the same, $i = 1,2,3,4$. As S_{5-i} and S_{5+i} could clearly form a pattern, all patterns could be packed by patterns with a guarantee of wasted space.

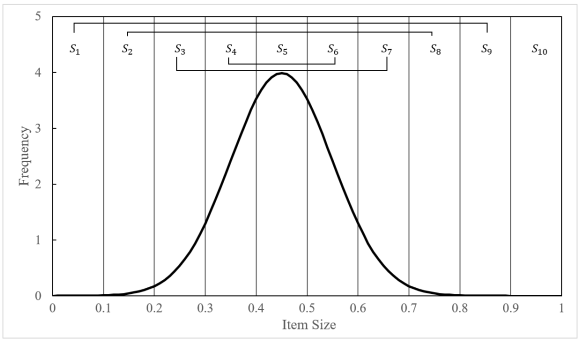


Figure 4. Example item size distribution (normal distribution with $\mu = 0.45, \sigma = 0.1$).

This feature would also enhance the packing of an approximately symmetrical distribution. Recall the previous example shown in Table 2. If we have accumulated some knowledge from history items and derive a prediction about the item size distribution, i.e., two S_4 items, two S_5 items, and two S_6 items, we could easily use the pattern $\{S_4, S_6\}$ and $\{S_5, S_5\}$ to guide the packing, which could generate an optimal solution.

3.3. Stability of Patterns in BPP

We deduce that patterns are stable in the solutions of BPP, so that a small error in the estimation of item distribution will only cause small deviation from optimal solutions.

Assume that the optimal solution of a BPP of n items $\{x_i\}, (i = 1, \ldots, n)$ is $OPT\{x_i\} = m$, where m is the number of bins used in the solution. Consider another BPP of n items $\{x'_i\}$ in which all items are same as $\{x_i\}$ except one item. Let m' be the number of bins used in the optimal solution of $\{x'_i\}$. It is straightforward to deduce that m' is at most $m + 1$ and at least $m - 1$.

Consider a solution to $\{x'_i\}$ which uses all the patterns of $OPT\{x_i\}$. We prove that the bins used in this solution are $m' + 1$ in the worst case.

Proof: Let x'_j be the item that is different from x_j and P be the pattern that contains x_j in $\{x_i\}$. If $x'_j \leq x_j$, x'_j, all other items can be packed into m bins according to the patterns of $OPT\{x_i\}$. So it is no more than $m' + 1$ because m' is at least $m - 1$.

If $x'_j > x_j$, the solution requires at most $m + 1$ bins, while there must be $m' \geq m$. Assume that $m' < m$ in this situation, e.g., $m' = m - 1$, this is conflict with $OPT\{x_i\} = m$ since $\{x_i\}$ can be packed into $m - 1$ bins. Thus, this solution is $m' + 1$ in the worst case.

Using patterns in online BPP depends on accurate estimation of item distribution, which should be updated dynamically in the process of bin packing.

4. A Pattern-Based Algorithm for Online BPP

4.1. Algorithm

We developed a pattern-based algorithm for online BPP. This algorithm generates a plan for pattern usage based on previous distributions and packs the items according to the plan. It includes three sub-algorithms: pattern generation, pattern update and packing. In this section, this algorithm is described and evaluated.

Firstly, the data structures are explained. This algorithm keeps recording the sizes of the incoming items into the distribution *dist* according to sections S. S could be seen as equally sized intervals ranging over $(0, 1]$. $N_{section}$ is the number of intervals, which could be used to configure the algorithm. For convenience, we use S_n to represent section n, $n = 1, 2, \ldots, N_{section}$. The range of S_n is $(\frac{n-1}{N_{section}}, \frac{n}{N_{section}}]$. An S_n item refers to an item with size in S_n. *dist* is a list of frequencies, with a size of $N_{section}$. $dist_n$ records the frequency of historical items with size in S_n.

patterns could be generated given $N_{section}$ and $N_{pattern}$, which is the number of patterns. *patterns* could be seen as a two-dimensional table, representing fixed combinations of differently sized items. The size of *patterns* is $N_{pattern} \times N_{section}$. $patterns_{ij}$ refers to the number of S_n item in the j-th pattern. $N_{pattern}$ could be used to configure the algorithm.

Based on *patterns* and *dist*, the pattern plan *plan* can be generated and updated. *plan* can be seen as a list of integer numbers. $plan_n$ is the remaining quota of $patterns_n$. Values in *plan* are decremented after patterns are applied, and updated regularly based on updated distance. The update frequency could be predefined with $N_{sampling}$, which determines the number of items processed between two *plan* updates.

Bin waiting queues w are a two-dimensional array of queues, with an identical size of *patterns*. w_{ij} is a queue of $patterns_i$ bins waiting for S_j items. Once a new bin is generated, it is pushed into the corresponding queues. One bin may be in several queues of w. Guided by w, every bin follows an unchangeable pattern until it is fully removed from all queues. Every time an item is packed into a bin, the corresponding occurrence pops out of the queue.

As sometimes an item cannot be matched to a pattern, Best-Fit would be used as a fallback option. B_b is a vector of all bins packed or to be packed by Best-Fit. It also receives bin packed to patterns regularly.

Secondly, the internal flow of this algorithm is explained. For clarity, we divide the entire algorithm into four parts: `Main` (Algorithm 1), `Pattern Generation` (See Appendix A: Algorithm A1), `Pattern Update` (See Appendix A: Algorithm A2) and `Packing` (See Appendix A: Algorithm A3). As can be seen in the `Main` function, this algorithm runs in iterations. One item is packed in one iteration. During the initialization stage, the patterns are generated, with other variables being empty or zero. In the first iteration, items are recorded into *dist* and processed by standard Best-Fit algorithm since *plan* is not yet generated and initially remains blank. Once the item counter reaches $N_{sampling}$, it is reset, and the *plan* is updated based on *dist*. Then, the algorithm starts packing with patterns. The *dist* continues to accumulate, on which the *plan* is updated every $N_{sampling}$ items.

Algorithm 1 Pattern-Based Online Bin Packing.

```
function MAIN($N_{sampling}$, $N_{section}$)
    initialize the distribution recorder $dist$
    $patterns \leftarrow$ GENERATE PATTERNS($N_{section}$, $N_{pattern}$)
    initialize the pattern plan $plan$
    initialize bin waiting queues $w$ with size of $N_{pattern} \times N_{section}$
    initialize a vector of Best-Fit bins $B_b$
    $itemCounter \leftarrow 0$
    while has next item $item$ do
        $itemCounter \leftarrow itemCounter + 1$
        $idx \leftarrow item / N_{section}$
        $dist_{idx} \leftarrow dist_{idx} + 1$
        PACK ITEM($plan$,$dist$,$patterns$,$B_b$,$w$,$item$)
        if $itemCounter = N_{sampling}$ then
            $itemCounter \leftarrow 0$
            $plan \leftarrow$ UPDATE PATTERN PLAN($dist$,$patterns$)
```

The `Pattern Generation` function generates available patterns. It searches for possible size combinations with the priority of producing patterns with large item sizes. It starts with a pattern with single largest item type and iteratively changes the type of the smallest item one at a time. If total size of the current pattern is equal to the bin capacity, it records the pattern. If total size of the current pattern is less than the bin capacity, it duplicates the current smallest item type. If the total size of the current pattern is larger than the bin capacity, it downsizes the currently smallest item type. If the smallest item type has reached zero, the item is removed from the pattern and the second smallest item type is downsized. It stops when the current pattern is empty or the recorded pattern count has reached $N_{patterns}$.

The `Update Pattern plan` function is to update the pattern plan based on *dist*. It scales down the historical item size distribution (*dist*) to a smaller distribution ($dist_p$) with size of $N_{sampling}$, and treat this as a prediction for the next $N_{sampling}$ items. Based on the prediction, it generates quotas for each pattern so that most items in $dist_p$ are prescheduled into a pattern. Note that the generation of quotas are sequential. Former patterns would be processed first. When a pattern is scheduled, the included items would be subtracted from $dist_p$.

The `Pack Item` function is to pack items into bins. After the item is classified into a section, it would firstly attempt to use patterns with large item sizes, i.e. low indices. When attempting to use a pattern, if there is a bin in the corresponding queue in w, the bin pops. If it cannot find a bin in the queue and there is still quota available in *plan* for this pattern, a new bin is created for packing and pushed into corresponding queues. If neither of two options above are available and this pattern does not include the section of this item, it attempts to use the next pattern. If no pattern is available, it falls back to use Best-Fit for packing with B_b. When the bin is found fully packed by a pattern, it is added to B_b.

4.2. Experiments

We have assessed the capability of this algorithm by coding with C++. First, we compare its performance with Best-Fit on Bin Usage for packing. We generate 10,000 medium-sized items with four distribution: a uniform distribution over $(3000, 6000]$, a triangular distribution with *mode* of 4500 over $(3000, 6000]$, a uniform distribution over $(2000, 6000]$ and a triangular distribution with *mode* of 4000 over $(2000, 6000]$. The algorithm is configured with $N_{pattern} = 100$, $N_{section} = 17$, and $N_{sampling} = 250$. The result (Figure 5) shows that, on some distributions where items have medium sizes, our algorithm has an advantage in bin usage compared to Best-Fit. Note that on other distributions or with other configurations, Best-Fit might have lower bin usage.

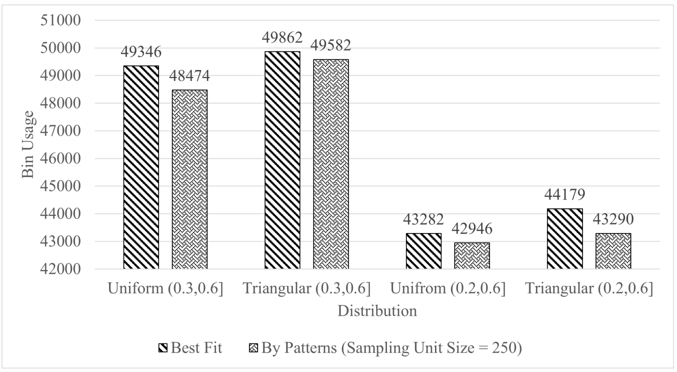


Figure 5. Performance comparison with Best-Fit by bin usage with sampling unit size 250 on uniform and triangular distribution over $(0.3, 0.6]$ and $(0.2, 0.6]$.

We compared the cause of the difference. Figure 6 shows the distribution of the wasted space of two algorithms over the uniform distribution over $(0.3, 0.6]$. It could be seen that in most of wasted space intervals, the frequency produced by our algorithm is less than Best-Fit. The overall trend is roughly a decrease in frequency as wasted space grows. The difference is that the frequency produced by Best-Fit shows a mild downward trend of frequency as waste space increases, while our algorithm gives a increase at first, followed by a sharp decrease. It means that although Best-Fit produces more accurate packing (with wasted space less than 0.05), our algorithm produces much more sub-accurate packing (with wasted space between 0.05 and 0.1). In this case, our algorithm sacrifices some possibility of packing tightest bins and largely enhances the possibility of packing bins with slightly lower tightness.

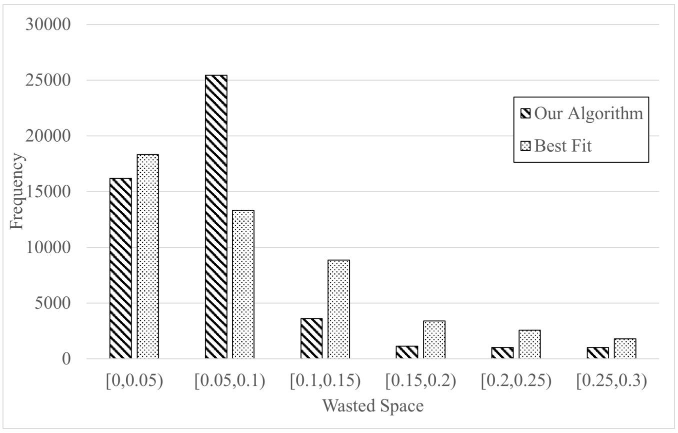


Figure 6. Performance comparison with Best-Fit by bin wasted space with sampling unit size 250 on uniform distribution over $(0.3, 0.6]$. Intervals with frequency less than 20 are ignored.

To further investigate the behavior of our algorithm, we compared the frequency and wasted space of each type of bin in the result (Figure 7). Recall that, in the algorithm, every bin is given a category at the beginning of its packing. In this case, there are four categories of bin: bins packed by pattern 20, bins packed by pattern 31, bins packed by pattern 46 and bins packed by Best-Fit as a fallback option. It could be seen that over $\frac{2}{3}$ of bins are packed by patterns. The most used patterns are pattern 31 and pattern 46, which produce the least average wasted space. Although Best-Fit are used as a fallback option, and produces bins with higher wasted space, it does not cover the advantage brought by patterns in performance.

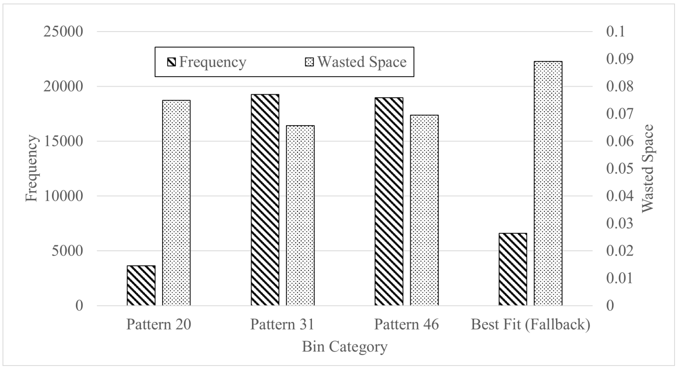


Figure 7. Comparison between each type of bin within our algorithm with frequency and wasted space with sampling unit size 250 on uniform distribution over $(0.3, 0.6]$.

We also compare its performance with Best-Fit on CPU running time. Under the experiment settings above, on the same environment (Table 6), we repeated tests 100 times and recorded the average CPU time values. The result is shown in Figure 8. Under such circumstances, the CPU time consumed by our algorithm is 13% to 66% lower than that of Best-Fit.

Table 6. Experiment environment.

CPU	**Intel® Core™ i7-12700KF@3.6 GHz**
RAM	16 GiB@4800 MHz
OS	Windows 11 Professional 21H2
SDK	Visual Studio 2019 (v142)
Build Conf.	Release x64

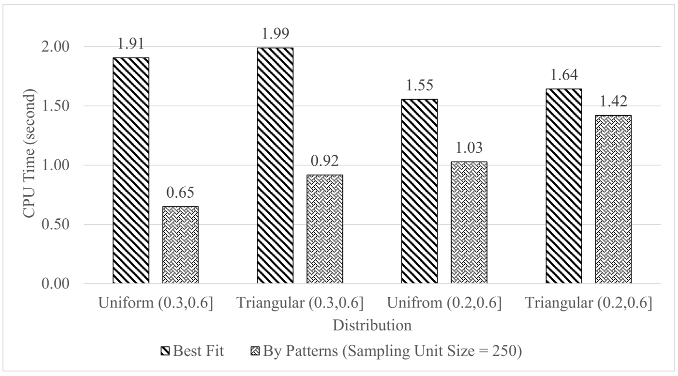


Figure 8. Performance comparison by CPU time with sampling unit size 250 on uniform and triangular distribution over $(0.3, 0.6]$ and $(0.2, 0.6]$.

In reality, the distribution of incoming item sizes is predictable based on historical data, but often changes over time. As our algorithm is based on distribution learning, distribution deviations would not change the result severely, especially when the changes are minor. An experiment designed to simulate this scenario, and to prove the feature of stability, is explained in Section 3.3. On different uniform distribution, some items are randomly selected and reassigned sizes in $(0, 1]$, which simulates changes in the distribution. With gradually deviated distributions, we record bin usage and internal pattern usage on different types of original distributions.

As can be seen in Figure 9, with the number of items with re-assigned sizes increasing, on different original distributions, the bin usage changes linearly towards the same static value, which is confirmed to be the bin usage over a uniform size distribution (0, 10,000]. This shows that a change to the original distribution would not severely degrade the performance of our algorithm. Instead, our algorithm would adapt to the change and deliver a reasonable result.

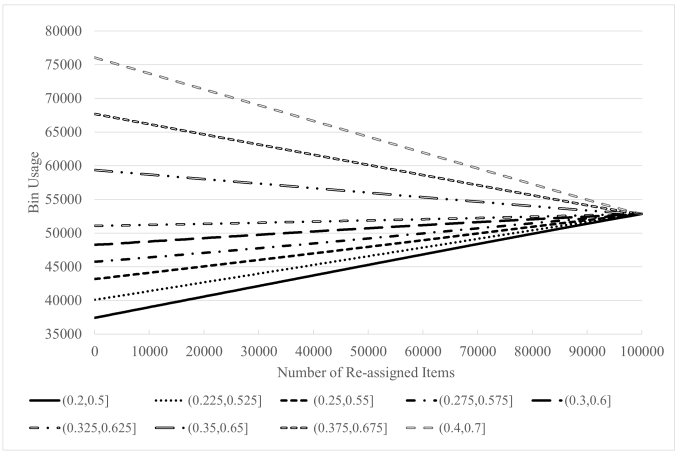


Figure 9. Bin usage with varied uniform distributions $(0.2, 0.5]$, $(0.225, 0.525]$, $(0.25, 0.55]$, $(0.275, 0.575]$, $(0.3, 0.6]$, $(0.325, 0.625]$, $(0.35, 0.65]$, $(0.375, 0.675]$, and $(0.4, 0.7]$ with sampling unit size 250; varied size over $(0, 1]$ for a random Item.

A minor change to the distribution causes an even more minor change to the internal pattern usage. Figure 10 shows that, with distribution slightly changed, the internal patterns would be changed as well. However, the change is minor. Figure 11 shows that on distribution $(0.3, 0.6]$, the deviation of the overall pattern usage does not exceed the varieties of items. With deviation of item size increasing from 0 to 20, the pattern usage fluctuates between -6 and 1. It proves that even with a varied item size, the usage of pattern is mild and stable.

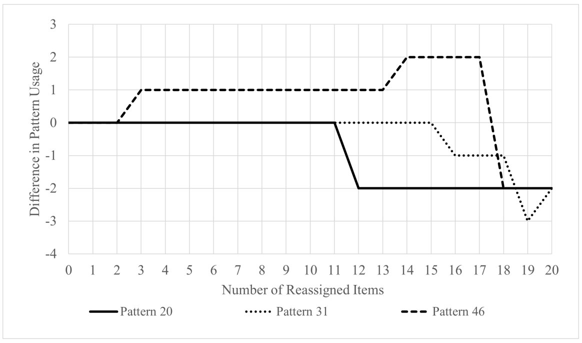


Figure 10. Usage of pattern 19, 30 and 45 with varied uniform distributions $(0.3, 0.6]$ with sampling unit size 250.

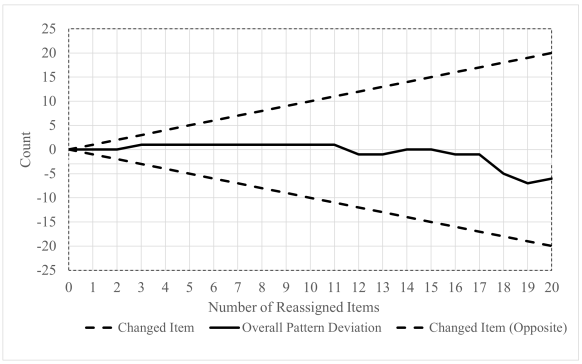


Figure 11. Overall pattern usage with varied uniform distributions $(0.3, 0.6]$ with sampling unit size 250.

5. Conclusions

In this research, we have analyzed the symmetry and stability of patterns in BPP and developed a pattern-based algorithm with parameterized symmetrical partitioning method for online BPP. The experimental results show that the proposed algorithm has a better performance for average bin usage, average wasted space and computational time compared with the heuristics of Best-Fit and First-Fit. The experiments also reveal that patterns still maintain stable in online BBP, even when the distribution varies slightly in an online BPP. This pattern based method provides a way to handle uncertainty in online BPP, which has the potential to be extended to general optimization problems with uncertainty.

The pattern-based algorithm depends on both estimation of item distribution and pattern selection. The proposed algorithm adopts heuristics to select patterns which do not compute the optimal patterns for a given distribution of items, despite the computational simplicity. This limitation may be overcome by using some meta-heuristics or machine learning methods to search pattern combinations for a given item distribution. A machine learning method may be efficient for pattern selection by computing near optimal pattern combinations with moderate computational complexity. This will be our future research focus.

Author Contributions: Software and investigation, B.L.; methodology and formal analysis, J.L.; supervision, project administration, R.B.; resources, R.Q.; validation, T.C. and H.J.; writing—original draft preparation, B.L.; writing—review and editing, J.L.; visualization, B.L.; funding acquisition, J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This study is funded by a NSFC (code 72071116) and Ningbo 2025 key technology projects (code 2019B10026, E01220200006).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Algorithm A1 Online Bin Packing Pattern Generation.

```
function GENERATE PATTERNS($N_{section}$, $N_{pattern}$)
    initialize $patterns$ with size of $N_{pattern} \times N_{section}$
    initialize $t$ with size of $N_{section}$
    $n_p \leftarrow 0$
    $t_0 \leftarrow N_{section}$
    $i \leftarrow 0$
    while $t_0 \neq 0$ & $n_p < N_{pattern}$ do
        $sum \leftarrow 0$
        for all $n$ in $t$ do
            $sum \leftarrow sum + n$
        if $sum = N_{section}$ then                                ▷ Record pattern
            for all $n$ in $t$ do
                $pattern_{n_p n} \leftarrow pattern_{n_p n} + 1$
            $n_p \leftarrow n_p + 1$
            $t_i \leftarrow t_i - 1$
        else if $sum < N_{section}$ then                           ▷ Pattern too small
            if $t_i \neq 0$ then                                   ▷ Duplicate
                $t_{i+1} \leftarrow t_i$
                $i \leftarrow i + 1$
            else                                                   ▷ Remove item
                $i \leftarrow i - 1$
                $t_i \leftarrow t_i - 1$
        else                                                       ▷ Pattern too large
            $t_i \leftarrow t_i - 1$                               ▷ Downsize
    return $patterns$
```

Algorithm A2 Online Bin Packing Pattern Update.

```
function UPDATE PATTERN PLAN($dist$,$patterns$)
    initialize $dist_p$
    $N_{dist} \leftarrow countSamples(dist)$
    for $i = 0$ to $N_{section}$ do                                ▷ Generate prediction
        $dist_{pi} \leftarrow dist_i \times (N_{sampling}/N_{dist})$
    initialize $plan$
    for $i = 0$ to $N_{pattern}$ do                                ▷ Generate pattern plan
        $usable \leftarrow false$
        $val_{min} \leftarrow \infty$
        for $j = 0$ to $N_{section}$ do
            if $patterns_{ij} \neq 0$ then
                $val \leftarrow dist_{pj}/patterns_{ij}$
                if $val < val_{min}$ then
                    $usable \leftarrow true$
                    $val_{min} \leftarrow val$
        if $usable$ then
            $plan_i \leftarrow val_{min}$
            for $j = 0$ to $N_{section}$ do
                $dist_{pj} \leftarrow dist_{pj} - val_{min} \times patterns_{ij}$
    return $plan$
```

Algorithm A3 Online Bin Packing Pack by Patterns.

```
function PACK ITEM($plan$,$dist$,$patterns$,$B_b$,$w$,$item$)
    $m \leftarrow floor[i/(C/N_{section})]$
    $packed \leftarrow false$
    initialize current bin $b$ as $null$
    for $i = 0$ to $N_{pattern}$ do
        if $patterns_{im} > 0$ then
            if QUEUE SIZE$(w_{im}) > 0$ then
                $b \leftarrow$ POP$(w_{im})$
            else if $plan_i > 0$ then
                $plan_i \leftarrow plan_i - 1$
                $b \leftarrow GetNewBin()$
                for $j = 0$ to $N_{section}$ do
                    for $h = 0$ to $patterns_{ij}$ do
                        PUSH$(w_{ij}, b)$

                POP$(w_{im})$
        if $b \neq null$ then
            PACK IN$(b, item)$
            if NOT EXIST$(w, b)$ then
                PUSH$(B_b, b)$
            $packed \leftarrow true$
    if $packed = false$ then
        BEST-FIT$(B_b, item)$
```

References

1. Johnson, D.S.; Demers, A.; Ullman, J.D.; Garey, M.R.; Graham, R.L. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM J. Comput.* **1974**, *3*, 299–325. [CrossRef]
2. Johnson, D.S. Fast algorithms for bin packing. *J. Comput. Syst. Sci.* **1974**, *8*, 272–314. [CrossRef]
3. Johnson, D.S. Near-Optimal Bin Packing Algorithms. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1973.
4. Yao, A.C.-C. New Algorithms for Bin Packing. *J. ACM* **1980**, *27*, 207–227. [CrossRef]
5. Coffman, E.G.; Csirik, J.; Galambos, G.; Martello, S.; Vigo, D. Bin Packing Approximation Algorithms: Survey and Classification. In *Handbook of Combinatorial Optimization*; Springer: New York, NY, USA, 2013, pp. 455–531.
6. Lee, C.C.; Lee, D.T. A simple on-line bin-packing algorithm. *J. ACM* **1985**, *32*, 562–572. [CrossRef]
7. Ramanan, P.; Brown, D.J.; Lee, C.C.; Lee, D.T. On-line bin packing in linear time. *J. Algorithms* **1989**, *10*, 305–326. [CrossRef]
8. Richey, M.B. Improved bounds for harmonic-based bin packing algorithms. *Discret. Appl. Math.* **1991**, *34*, 203–227. [CrossRef]
9. Seiden, S.S. On the online bin packing problem. *J. ACM* **2002**, *49*, 640–671. [CrossRef]
10. Balogh, J.; Békési, J.; Galambos, G. New lower bounds for certain classes of bin packing algorithms. *Theor. Comput. Sci.* **2012**, *440–441*, 1–13. [CrossRef]
11. Heydrich, S.; van Stee, R. Beating the Harmonic Lower Bound for Online Bin Packing. In Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), Rome, Italy, 12–15 July 2016; pp. 41:1–41:14. Available online: http://drops.dagstuhl.de/opus/volltexte/2016/6321 (accessed on 10 December 2021).
12. Silva-Gálvez, A.; Lara-Cárdenas, E.; Amaya, I.; Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C. A Preliminary Study on Score-Based Hyper-heuristics for Solving the Bin Packing Problem. In *Pattern Recognition*; Lecture Notes in Computer Science; Springer Science+Business Media: Berlin/Heidelberg, Germany, 2020; Chapter 30, pp. 318–327.
13. Dunke, F.; Nickel, S. A data-driven methodology for the automated configuration of online algorithms. *Decis. Support Syst.* **2020**, *137*, 113343. [CrossRef]
14. Gupta, V.; Radovanović, A. Interior-Point-Based Online Stochastic Bin Packing. *Oper. Res.* **2020**, *68*, 1474–1492. [CrossRef]
15. Ross, N.; Keedwell, E.; Savic, D. Human-Derived Heuristic Enhancement of an Evolutionary Algorithm for the 2D Bin-Packing Problem. In *Parallel Problem Solving from Nature—PPSN XVI*; Springer: Cham, Switzerland, 2020; pp. 413–427.
16. Zhao, H.; She, Q.; Zhu, C.; Yang, Y.; Xu, K. Online 3D bin packing with constrained deep reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtually, 2–9 February 2021; AAAI Press: Palo Alto, CA, USA, 2021, Volume 35, pp. 741–749.