A Data-Driven Approach for Multi-level Packing Problems in Manufacturing Industry

Lei Chen
Huawei Noah's Ark Lab
lc.leichen@huawei.com

Xialiang Tong
Huawei Noah's Ark Lab
tongxialiang@huawei.com

Mingxuan Yuan
Huawei Noah's Ark Lab
yuan.mingxuan@huawei.com

Jia Zeng
Huawei Noah's Ark Lab
zeng.jia@huawei.com

Lei Chen
Hong Kong University of Science and Technology
leichen@cse.ust.hk

ABSTRACT

The bin packing problem is one of the most fundamental optimization problems. Owing to its hardness as a combinatorial optimization problem class and its wide range of applications in different domains, different variations of the problem are emerged and many heuristics have been proposed for obtaining approximate solutions.

In this paper, we solve a ***M***ulti-***L***evel ***B***in ***P***acking (***MLBP***) problem in the real make-to-order industry scenario. Existing solutions are not applicable to the problem due to: 1. the final packing may consist multiple levels of sub-packings; 2. the geometry shapes of objects as well as the packing constraints may be unknown. We design an automatic packing framework which extracts the packing knowledge from historical records to support packing without geometry shape and constraint information. Furthermore, we propose a dynamic programming approach to find the optimal solution for normal size problems; and a heuristic multi-level fuzzy-matching algorithm for large size problems. An inverted index is used to accelerate strategy search. The proposed auto packing framework has been deployed in Huawei Process & Engineering System to assist the packing engineers. It achieves a performance of accelerating the execution time of processing 5, 000 packing orders to about 8 minutes with an average successful packing rate as 80.54%, which releases at least 30% workloads of packing workers.

CCS CONCEPTS

• **Applied computing** → **Industry and manufacturing**; *Decision analysis*; • **Computing methodologies** → *Planning and scheduling*; *Search methodologies.*

KEYWORDS

Multi-level bin packing; Data-driven; Dynamic programming; Fuzzy matching

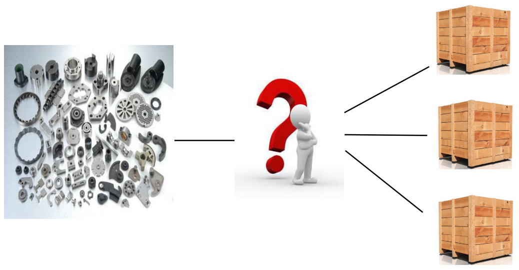


Figure 1: A real-life packing scenario

ACM Reference Format:
Lei Chen, Xialiang Tong, Mingxuan Yuan, Jia Zeng, and Lei Chen. 2019. A Data-Driven Approach for Multi-level Packing Problems in Manufacturing Industry. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330708

1 INTRODUCTION

Making superior packing plans efficiently is of crucial importance in implementing seamless business logistics, which is one of the key elements in keeping pace with customer demands and provides a competitive edge against other organizations. To this end, enterprises, especially in the fields of courier and manufacture, employ many workers specialized in doing packing jobs. Humans usually do packings in a trial-and-error process. Thus, manual packing highly depends on the experience that the workers have and their knowledge about the packing criteria and estimating the sizes of the items. An experienced packing worker may need 1 or 2 trials to successfully configure a packing plan for an order. Moreover, the training period for a job-qualified packing worker could take more than 3 months. Enabling auto-packing is therefore of great significance to improve the efficiency of the logistics in enterprises, which further helps to improve their profits, maintain their competitive edges and build good consumer relations.

The bin packing problem is one of the most fundamental problems in optimization. Besides the application in logistics transportation [19], it has wide applications in domains including job scheduling, financial stock decisions, computer memory management, *etc.* Due to various problem settings and constraints in different applications, many variants of the bin packing problems have been studied. The general form of the problem is to consider sets of items and item-holding objects called bins and aim to group items in a way that they all fit into a number of bins, while some certain constraints hold and a cost function is optimized.

In this paper, we solve a multi-level bin packing problem in real-life scenarios, where the bins are divided into multiple levels according to their materials and volumes. Lower level bins can be packed into higher level ones and items can be packed into each level of the bins. The task of MLBP is to pack a set of items into a set of the highest level bins, namely *SPU* (*Standard Product Unit*) bins, where inside each such a bin, there could exist lower level bins, namely *SKU* (*Stock Keeping Unit*) bins that are used to protect the items or group the items together. In addition to the more complex packing structure, two more challenges exist in real-life packing scenarios. One is that, unlike the assumption in most literatures, the exact space that an item occupies in a bin is usually unknown. This is because that items could be of irregular shape (see Fig. 1 as an example) and accessories such as molded foam and bubble wrap are usually packed together with the items to keep them safe from damage. The other challenge is that, rather than the cost, more factors need to be considered in real-life packings, such as various rules and restrictions on the accommodation of items , balance of the weight of each bin, customer preferences, *etc.*

Taking all the aspects of the real-life packing problem into consideration and solve it by a traditional algorithm is impossible. Fortunately, like the packing pipeline in Huawei, historical packing data by specialized workers may exist, which contains a wealth of information about the packing constraints and objectives. Taking advantage of the historical manual packing data, we study the real-life MLBP problem from a data-driven perspective and propose a light-weight approach to solve it. Moreover, we apply crowdsourcing to leave packing problem instances whose packing plans cannot be derived from historical records, *e.g.,* packing brand new items, to specialized workers for manual packing, where the solution plan can then further enrich the historical packing database.

The contributions made in this paper are summarized as follows:

- To the best of our knowledge, we are the first to study the packing problem from a data-driven perspective, where we use historical packing records to guide future packings.
- We solve the packing scenario in real applications, where the final packing consists of multiple levels of sub-packings. The solution is able to integrate the packing requirements and objectives that human experts consider and is able to support packings for items with unknown sizes.
- We propose a dynamic programming algorithm to find optimal solutions for the packing problem with normal sizes. And we propose a heuristic multi-level matching algorithm to deal with problems with large sizes more efficiently.
- We reduce the historical record database (search space) by processing a skyline query and we build an inverted index to accelerate the processing.
- We deploy the proposed algorithms on the packing line in Huawei Process & Engineering System. The results demonstrate the efficiency and effectiveness of the proposed algorithms, by achieving a performance of accelerating the execution time of processing $5,000$ packing orders to about 8 minutes with an average successful packing rate as 80.54%, that is, a releasing of more than 30% packing workers.

2 RELATED WORK

The bin packing problem is one of the earliest problems concerned in the literature of operations research. It is one of the well-known combinatorial NP-hard problems [1] in computational complexity theory. The bin packing is usually defined as packing a set of items into a number of item-holding objects named as bins such that the dimension and capacity constraint of each bin holds, while a cost function is optimized. The cost function could be minimizing the number of used bins, the wasted bin space or the cost of bins, *etc.* The problem has many variations, which can be classified into different categories from different perspectives [5]. According to the number of dimensions, the bin packing problems can be classified into 1D [7, 17], 2D [2] and 3D [10, 13, 18]. Depending on whether the items and bins are known in advance, there are on-line [4] and off-line [13, 16] problems. It can also be categorized into single-sized [15] and variable-sized problems [6, 9] by the types of bins.

Different algorithms have been proposed to solve bin packing problems in different settings. The exact algorithms usually model bin packing problems as integer programming and then solve them using branch and bound techniques. As a set of NP hard problems [1], the exact algorithms are only applicable in problems with small sizes [3, 18]. More algorithms [4, 9] aim to find sub-optimal solutions in short response time using approximate algorithms, which typically lie in the following categories. One class of the algorithms is to design some *greedy heuristic* strategies, such as Next-fit, Best-fit [11], the wall building algorithm [7], *etc.* Another class of the algorithms is to use *meta-heuristic* to guide the search of near-optimal solutions, such as Tabu Search [14] and Genetic Algorithm [8, 12]. More recently, there emerges a class of algorithms that adapt *artificial intelligence* and *machine learning* techniques to solve different bin packing problem instances [10, 13, 15, 17]. The AI/ML techniques are usually used in the form of training classification/regression models for the selection of different heuristic strategies or measuring the quality of a candidate solution.

However, real-life packing problems are much more complex than those studied in the literature. Multiple levels of sub-packings could be required. The exact item volumes, packing constraints and optimization objectives cannot be explicitly expressed. No existing work can be directly used. Fortunately, specialized packing workers can take the packing criteria (e.g., accommodation rules, maximum weight constraint, fragile item packing rules, customer preferences, etc.) into consideration and make certain-degree optimal packing plans. In this paper, we consider the case that manual historical packing data is available. This is applicable in many scenarios as long as packing is not a brand new business for them. The constraints and objectives of the real-life packing problems are veiled in the historical data. Instead of making packing plans from scratch, we do packing for future packing orders based on the historical packing records. To the best of our knowledge, we are the first to study the bin packing problem from a data-driven perspective.

3 PROBLEM FORMULATION

3.1 Multi-level Bin Packing Problem

Consider an item set $\mathcal{I}$ and a bin set $\mathcal{B}$. The bins in $\mathcal{B}$ are divided into two categories, *i.e., SPU* bins and *SKU* bins. An *SPU* bin is known as a *Standard Product Unit*. An *SPU* is the most outside level

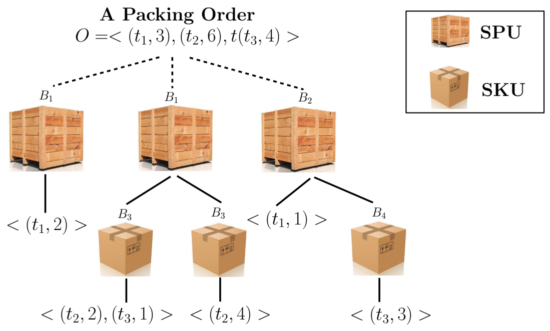


Figure 2: An Example Multi-Level Bin Packing

a packing. These bins can be directly shipped. *SKU* bins are referred as *Stock Keeping Unit*s. They are middle levels of a packing. An *SKU* bin is used to protect items that cannot be directly put into *SPU*s; or grouping a set of items together in the packing. An *SKU* bin needs to be packed into *SPU* bins for shipment. For the ease of presentation, we consider the MLBP problem of 2 levels, meaning that there exists no *SKU* bins packed in another *SKU* bin. Note that the algorithms and optimizations proposed in this paper can be easily adapted to support packing cases with more levels.

Definition 1. ***Packing Order.*** *A packing order lists a set of items and the number of each item to be packed. It is defined as $O =< (t_i, q_i)|i \in N^+ >$, where t_i denotes an item and q_i denotes its quantity.*

Definition 2. ***Packing Plan.*** *A packing plan P describes the detailed packing configuration for a packing order O. It determines how the items are grouped and packed into SPU/SKU bins.*

A packing plan for a given packing order is a set of trees, where the root node of each tree is an *SPU* bin, the internal nodes are *SKU* bins, and the leaf nodes are items.

Example 1. *Fig. 2 illustrates an example of a packing plan for a packing order. The order asks to pack 3 of the item t_1, 6 of the item t_2 and 4 of the item t_3, respectively, where B_1, B_2 are two different SPU bins and B_3, B_4 are two types of SKU bins. The packing plan for the order is of tree structures. As we can see, the plan uses 3 SPU bins, i.e., two B_1 bins and one B_2 bin. The first B_1 SPU bin directly packs 2 t_1 items. The second B_1 bin packs 2 B_3 SKU bins, where one B_3 bin is packed with 2 t_2 items and 1 t_3 item and the other packs 4 t_2 items. The last SPU bin B_2 packs 1 t_1 item and a B_4 SKU bin, where B_4 is packed with 3 t_3 items.*

Definition 3. ***Packing Cost.*** *The packing cost defines the total cost of a packing plan. Given a packing plan P, the packing cost of P could be measured by a cost function, e.g., a cost function considering the total prices of the used bins can be defined as follows: $Cost(P) = \Sigma_{\forall b \in SPU \cup SKU} price(b)$.*

Definition 4. ***Multi-level Bin Packing Problem.*** *Given a packing order O, the MLBP problem is to find the packing plan P for O with the minimum cost according to a cost function while all packing constraints are satisfied.*

Theorem 1. *The multi-level bin packing problem is NP-hard.*

3.2 Problem Analysis

Given a packing order $O =< (t_i, q_i)|i \in N^+ >$, to answer a MLBP problem, a straight-forward method is to divide the packing order into several subsets and test if each subset can fit into *SPU/SKU* bins; then among all different divisions that all subsets satisfy all constraints, return the one with the minimal cost *w.r.t.* a cost function. This approach is obviously inapplicable, since it needs to exhaustively test all combinations of the packing order. Moreover, the test of if a set of items can fit into a bin may incur the measurement of item sizes. The order/position of each item needs to be considered and even a real packing trial may be needed to test the fitness. Meanwhile, the consideration of accommodation rules, balancing, *etc*, could make it more complex. Taking all these aspects into consideration makes the packing problem unsolvable.

An obvious thoughtless in traditional packing algorithms is that the existing packing records which contain a wealth of knowledge are not used. We consider the packing problem from a *data-driven* perspective.

Definition 5. ***Packing Record.*** *A packing record is a packing plan for a packing order in the history.*

We remark that the detailed packing knowledge, such as the location, the orientation and the protecting accessories of each item, can also be stored in the packing record. As for a MLBP problem, a packing record can be further divided into three level, *i.e.*, *Order level*, *SPU level* and the *SKU level*. See Fig. 2 as an example. In *Order level*, we can observe the whole packing plan for the packing order $< (t_1, 3), (t_2, 6), (t_3, 4) >$. In *SPU level* level, we know that the *SPU* bin B_1 is able to pack $< (t_1, 2) >$ or $< (t_2, 6), (t_3, 1) >$, and the *SPU* bin B_2 is able to pack $< (t_1, 1), (t_3, 4) >$. And in the *SKU level*, we observe that, the *SKU* bin B_3 is able to pack $< (t_2, 2), (t_3, 1) >$ or $< (t_2, 4) >$, and the *SKU* bin B_4 is able to pack $< (t_3, 3) >$.

Theorem 2. *Given a historical packing plan P regarding a packing order $O =< (t_i, q_i)|i \in N^+ >$, a new packing order $O' =< (t'_i, q'_i)|i \in N^+ >$ can definitely be packed with P if $\forall t'_i \in O', t'_i \in O \bigwedge q'_i \le q_i$.*

Theorem 2 inspires a way to determine if a set of items can fit into a bin without explicitly knowing the item sizes and packing constraints, *etc.* The rationality of considering the historical packing records as a guidance can be supported by the following aspects. First, the packing plans in the history are made by specialized packing workers. We could make the assumption that these historical packing plans are *"optimal"* for the historical packing orders, where the capacity constraint, accommodation rules and other hidden constraints that we don't realized are satisfied. Second, in industry scenarios, large numbers of future orders have identical or similar orders as well as their packing plans exist in the history. Only small proportion of orders are brand new. For these new orders, we pass them to specialized packing workers for manual packing. A nice property is that, such a manual packing only needs to be processed once, and its result can be inserted to the historical packing record database to further guide the future packings.

4 A DYNAMIC PROGRAMMING APPROACH

Theorem 2 introduces a way to determine whether a set of items can be packed into a bin or with a packing plan. However, the baseline algorithm still needs to try all possible subsets, which is obviously

Algorithm 1 State Encoder and Decoder

```
    Function Encode(Packing Order $O$, $multi$-d State $\vec{s}$
1:    $num \leftarrow \prod_{i=1}^{|O|}(O.q_i + 1)$
2:    $s \leftarrow 0$
3:    for $i = 1$ to $|O|$ do
4:        $num \leftarrow num/(O.q_i + 1)$
5:        $s \leftarrow s + num * \vec{s}.q_i$
6:    return $s$
    Function Decode(Packing Order $O$, 1-d State $s$)
7:    $num \leftarrow \prod_{i=1}^{|O|}(O.q_i + 1)$
8:    for $i = 1$ to $|O|$ do
9:        $num \leftarrow num/(o.q_i + 1)$
10:       $\vec{s}.q_i \leftarrow s\ div\ num$
11:       $s \leftarrow s\ mod\ num$
12:   return $\vec{s}$
```

computationally intensive. In this section, we introduce a dynamic programming algorithm towards solving the MLBP problem.

4.1 Dynamic Programming

The main idea of dynamic programming is to build up the solution for a complex problem from the answers to its sub problems. Since the detailed pack plan about *SKU* bins can be inherited from that of their father nodes, *i.e.*, *SPU* bins, in the packing plan tree. We now focus on how to divide a packing order into *SPU*s, such that the total cost of the final packing plan is minimized.

After a packing order $O =< (t_1, q_1), (t_2, q_2), ..., (t_m, q_m) >$ is specified, we can focus only on the items that in O then. As such, the packing order O can be represented as a m-dimensional vector, *i.e.*, $\vec{s_O} =< O.q_1, O.q_2, ..., O.q_m >$, meaning that it asks to pack the corresponding numbers of items $t_1, t_2, ..., t_m$, respectively. Similarly, by ignoring the detailed packing tree structures, each existing *SPU* level packing record p can be represented *w.r.t.* the packing order as a map from a *SPU* bin to a quantity vector, *i.e.*, $(B, < q_1, q_2, ..., q_m >)$, meaning that the *SPU* bin B is able to pack q_1, q_2,..., q_m numbers of t_1, t_2,..., t_m, respectively. Let $f(\vec{s})$ denote the minimum cost of packing the items in $\vec{s}$. The Bellman Equation for computing $f(\vec{s})$ can be written as follows:

$$f(\vec{s}) = \min_{p \in D}(Cost(p) + f(< \max(\vec{s}.q_1 - \vec{p}.q_1, 0), \dots, \max(\vec{s}.q_m - \vec{p}.q'_m, 0) >)), \tag{1}$$

where $p = (B, \vec{p})$, D is the whole database of the *SPU* level packing records and $Cost(p)$ measures the cost of the packing plan. The cost can be measured from the perspective of total prices, number of used bins, or wasted space, etc. And the cost can be considered as the *SPU* bin's cost adding the costs of the *SKU* bins in the subtree.

As the number of different items varies in different packing orders, to make the algorithm generally applicable with different numbers of item types, we compress the multi-dimensional states into 1-dimension ones. The goal is to find the packing plan for the state of the packing order, *i.e.*, $\vec{s_O} =< O.q_1, O.q_2, ..., O.q_m >$. Thus, we only need to consider the previous states of $\vec{s_O}$, *i.e.*, $\{< q_1, q_2, ..., q_m > | 0 \le q_i \le O.q_i, 0 \le i \le m\}$, where exists in total $\prod_{i=1}^{|O|}(O.q_i + 1)$ states. The compression of the multi-dimension states into 1-dimension is actually to number the states from 0 to $\prod_{i=1}^{|O|}(O.q_i + 1) - 1$. The pseudo code of encoding the multi-dimensional states is given in Function *Enocde*(), Algorithm 1. The

Algorithm 2 Dynamic Programming

```
INPUT: Historical $SPU$ level Packing Plan Database $D$,
       Packing Order $O =< (t_1, q_1), (t_2, q_2), \dots, (t_m, q_m) >$
OUTPUT: The packing plan for $O$
1: $\vec{s_O} \leftarrow < q_1, q_2, \dots, q_m >$
2: $num \leftarrow \prod_{i=1}^{|O|}(q_i + 1)$
3: initialize $f[i]$, $\forall 1 \le i < num$, with a maximum default value $\hat{C}$
4: $f[0] \leftarrow 0$
5: for $s = 0$ to $num - 1$ do
6:     if $f[s] < \hat{C}$
7:         $\vec{s} \leftarrow Decode(O, s)$
8:         for each $SPU$ packing plan $(B, \vec{p})$ in $D$
9:             $\vec{s'} \leftarrow \vec{s} + \vec{p}$
10:            if $\vec{s'} \not\le \vec{s_O}$
11:                $\vec{s'}.q_i \leftarrow \min(\vec{s'}.q_i, \vec{s_O}.q_i), 0 < i \le m$
12:            $s' \leftarrow Encode(O, \vec{s'})$
13:            if $f[s] + Cost(\vec{p}) < f[s']$
14:                $f[s'] \leftarrow f[s] + Cost(\vec{p})$
15:                $h[s'] \leftarrow s$
16:                $b[s'] \leftarrow B$
17: $s \leftarrow num - 1$
18: $ans \leftarrow set()$
19: while $s \ne 0$
20:     $\vec{p} \leftarrow Decode(O, s) - Decode(O, h[s])$
21:     $ans.add((b[s], \vec{p}))$
22:     $s \leftarrow h[s]$
23: return $ans$
```

decoding works reversely and the pseudo-code is also given in Function *Decode*(), Algorithm 1.

With encoding/decoding of the states, the Bellman Equation of the dynamic programming algorithm can be rewritten as:

$$f(s) = \min_{p \in P}(f(Decode(s) - \vec{p}) + Cost(p)) \tag{2}$$

Based on all the above, we give the dynamic programming algorithm in Algorithm 2. It takes as input the historical packing plan database D and a query packing order O, and it returns the packing plan for O, *i.e.,* the assignment of items to each of the *SPU* bins that achieves the minimum total cost. First, we compute the state space, and initialize the cost of the start state as 0 and that of the other states with a maximum default value (Line 2–4). We present a forward DP implementation of the problem. We traverse all the states in ascending order (Line 5). For each state, we scan each historical *SPU* record $(B, \vec{p})$ to see if we next use the *SPU* packing plan can achieve a smaller cost for the future state. A smaller cost can always be achieved by sufficiently use the current *SPU* bin B, *i.e.,* packing as many items as the packing record into the *SPU* bin. Thus, we only need to check the future state $\vec{s'} \leftarrow \vec{s} + \vec{p}$ (Line 9). Since the dynamic programming algorithm considers previous states of $\vec{s_O}$, we make sure that every dimension of $\vec{s'}$ is $\le$ that of $\vec{s_o}$ (Line 10–12). It is reasonable in doing so, as some of the bins can be partially packed. For the convenience of backtracking the *SPU* bins and their packing configurations, when update a better cost for the future state $\vec{s'}$, we also record its previous state $\vec{s}$ that leads to the better cost (Line 15) and the bin used to pack items represented in $\vec{s'} - \vec{s}$ (Line 16). With all these, after the forward DP process, we can easily traceback the path that leads to the smallest cost and obtain the assignment of items to each bin (Line 17–22).

Theorem 3. *Let $O =< (t_1, q_1), (t_2, q_2), ..., (t_m, q_m) >$ denote a packing order and let D be the database of SPU level packing records*

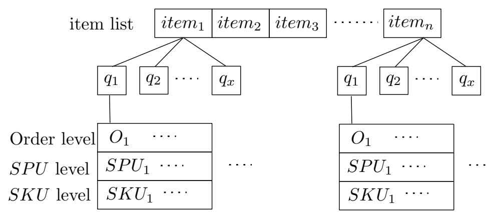


Figure 3: Structure of the inverted index

regarding the items in O. The time complexity of the dynamic programming algorithm is $O(|D| \cdot \prod_{i=1}^{|O|}(q_i + 1) \cdot |O|)$.

4.2 Optimizations

4.2.1 Inverted Index. After receiving a packing order $O =< (t_1, q_1), (t_2, q_2), ..., (t_m, q_m) >$, we actually only need to consider those packing records *w.r.t.* the items in O, *i.e.*, $D' = \{p|p \in D \wedge (\exists t \in O, t \in p)\}$, rather than the whole *SPU* level packing record database D. Instead of scanning the whole database, we build an inverted index to retrieve the related records in D' more efficiently. Fig. 3 illustrates the structure of the index. There is a 3-level "tree" for each item in the index, where the first level is the item itself, the second level refers to different quantities that the item packed in different records, and the third level contains 3 posting lists denoting that the corresponding number of the items are packed in packing records in *Order*, *SPU*, and *SKU* levels, respectively. To avoid duplicated storage, the posting lists only store links that point to the addresses of the corresponding packing records in each level. Hash is used to directly locate the index tree of a specific item. The inverted index can support both in-memory and disk-resident implementations.

With such an index, the related *SPU* level record set D' can be retrieved as the union set of the records in *SPU* level posting lists for each item in the query packing order O, where the maximum time complexity is $O(|O| \cdot |D'|)$.

4.2.2 Skyline. Intuitively, given two packing records p_1 and p_2, if p_1 packs more items than p_2 but with a smaller cost, we can say that the packing record p_1 is better than p_2. Using both p_1 and p_2 as guidances can achieve no better packing plans for a query packing order than just using p_1. We formally define this "better" as follows.

Definition 6. ***Dominance Relationship.*** *Given two packing records* $p_1 = (B_1, \vec{p_1})$ *and* $p_2 = (B_2, \vec{p_2})$, *we say that the packing record* p_1 *dominates* p_2 *if* $Cost(p_1) < Cost(p2) \wedge \forall i \in [1, m], \vec{p_1}.q_i \geq \vec{p_2}.q_i$; *or* $Cost(p_1) = Cost(p2) \wedge \forall i \in [1, m], \vec{p_1}.q_i \geq \vec{p_2}.q_i \wedge \exists i \in [1, m], \vec{p_1}.q_i > \vec{p_2}.q_i$.

Definition 7. ***Skyline Packing Records.*** *Given a packing record database D, the skyline packing records in D are those records that are not dominated by any other record in D.*

Given a packing record O, we further reduce the packing record database as the skyline records of its related record set D'.

4.2.3 Greatest Common Divisor. The dynamic algorithm explores each previous state of $\vec{s_O}$. For each state, a scan of related historical packing records is needed. The previous optimizations accelerate the retrieval of the record database and reduce its size. We now present that the number of states to visit can also be reduced.

Let $\vec{gcd}$ denote a m-dimensional vector, where $\vec{gcd}.q_i$ is the greatest common divisor of $\{\vec{p}.q_i|\vec{p} \in D'\}$. We reduce the states by replace each dimension of each packing record vector $\vec{p}$ with $\frac{\vec{p}.q_i}{\vec{gcd}.q_i}$; and each dimension of the query packing order vector $\vec{s_O}$ is updated as $\lceil \frac{\vec{s_O}.q_i}{\vec{gcd}.q_i} \rceil$.

Orders have similarities in real-life applications. Some item may be required to be sold or packed with the smallest batch quantity. The greatest common divisor of the quantities of an item in historical records can be greater than 1 or even much larger. Thus, the optimization is able to reduce the states to visit.

5 A MULTI-LEVEL FUZZY MATCH ALGORITHM

While the optimizations make the dynamic programming algorithm more efficient, it still has the chance to visit $\prod_{i=1}^{|O|}(q_i + 1)$ states, where for each state, a scan of the related historical packing records needs to be processed. Algorithms with such a time complexity will not be able to well support on-line services for large-scale applications. We proceed to propose a multi-level fuzzy matching algorithm that is able to find an approximate solution with less response time for a given query packing order while guarantees the space wastage ratio of each bin is within a given threshold ϵ.

5.1 Volume Estimation

To better estimate how much space is wasted in a packing plan, we estimate the volume of each item from the historical *SPU* and *SKU* level packing records.

Definition 8. ***Space Wastage Ratio.*** *Given an SPU/SKU level packing record* $p = (B, < (t_1, q_1), (t_2, q_2), ..., (t_{|p|}, q_{|p|}) >)$, *the space wastage ratio* ϵ *of the packing plan is defined as* $\epsilon = 1 - \frac{\sum_{i=1}^{|p|} t_i.v * q_i}{B.v}$, *where* $t_i.v$ *and* $B.v$ *denote the volumes of items and bin, respectively.*

The processing of estimating item volumes works as follows:

(1) We first find the set P of all *SPU*/*SKU* level packing records.
(2) An obvious observation is for a pack plan p, the total volume of the items in p is $\leq$ the volume of the bin $p.B$. We assume that the historical packing records waste as little space as possible, *i.e.*, the space wastage ratio of each bin is minimized. Thus, we can model the estimation of $t.v$ as the following linear programming problem.

$$minimize : \epsilon_1 + \epsilon_2 + \ldots + \epsilon_{|P|}$$

$$\forall p_j \in P, (1 - \epsilon_j) * p_j.B.v \leq \sum_{i=1}^{|p_j|} |p_j.t_i.v * p_j.q_i| \leq p_j.B.v \tag{3}$$

where $p_j.B$ denote the bin used in the pack plan p_j; $p_j.t_i$ and $p_j.q_i$ denote the item and its quantity in p_j, respectively.
(3) We solve the linear programming problems and get the estimated volume $t_i.v$ and each bin's space wastage ratio ϵ_j.

Note that the LP problem in Eqn. 3 is unsolvable when there are not enough related packing records. We compute the solvable problems first; and substitute the estimated volumes into other unsolved LP problems to reduce the number of variables. The process continues until all LP problems are solved or no other item's volume can be estimated.

Algorithm 3 Multi-Level Fuzzy Match Algorithm

INPUT: a query packing order $O = <(t_1, q_1), (t_2, q_2), \ldots, (t_m, q_m)>$; Historical Order, *SPU*, and *SKU* level packing plan databases D_{order}, D_{SPU}, and D_{SKU}, respectively.
OUTPUT: the packing plan for O

```
1: $p \leftarrow Matching_{order}(O, \ldots)$
2: if $p = null$
3:     $(p, O') \leftarrow Matching_{spu}(O, \ldots)$
4:     while $O' \neq null$ do
5:         $(p', O') \leftarrow Matching_{sku}(O', \ldots)$
6:         $p \leftarrow p \cup p'$
7: return $p$
```

5.2 Overview

With the estimated item volumes, we present the proposed multi-level fuzzy matching algorithm. The basic idea works as follows. Given a packing order $O = <(t_1, q_1), (t_2, q_2), ..., (t_m, q_m)>$, the algorithm first checks if there exists a historical order level packing plan p that can pack all the items and the space wastage ratio of each used bin is smaller than a given threshold ϵ. If such order level packing plans exist, we return the plan with the smallest cost and its detailed packing configuration as the packing plan for O. Otherwise, the algorithm checks if there exists an *SPU* level packing plan can pack all or part of the items in O, such that the *SPU* bin's space wastage ratio is $\leq \epsilon$. If such *SPU* level packing plans exist, we choose the one with the smallest cost and pack as many items as the plan can hold. The check and assignment of the *SPU* level packing plan continue until there exists no such an *SPU* level packing plan; or there remains no unpacked items. If after processing the *SPU* level matching, there still exist unpacked items. We continue to *SKU* level matching in a similar way to that of *SPU* level matching until all items are packed.

The pseudo code of the framework is given in Algorithm 3.

5.3 Matching by Orders ($Matching_{order}()$)

The order level matching is simple and straightforward. It is obviously that if there exists a historical packing plan p' for an order O' that is identical to the query packing order O, we can directly pack O with p'. More generally, given a packing order $O = <(t_1, q_1), (t_2, q_2), ..., (t_m, q_m)>$, if there exists a historical packing plan p' for an order O' that $\forall t_i \in O, t_i \in O' \wedge O'.q_i \geq O.q_i$, the packing plan p' can definitely packing all items in O. In order level matching, among all such packing records, we choose the one with the smallest cost and the space wastage ratio of each *SPU* bin is smaller than the given threshold ϵ. The pseudo code of the order level matching is given in Algorithm 4. Instead of accessing the whole database, we first retrieve each set $record_i$ of order level packing records containing more quantity of item t_i in O via the inverted index (Line 1–4). The set R of packing records that can pack all items in O are then obtained by joining all $record_i$. It returns *null* if R is empty; otherwise, the plan p' with the least cost and satisfies the space wastage constraint is returned.

5.4 Matching by SPUs ($Matching_{spu}()$)

Recall that the goal of the MLBP problem is to pack the items in an packing order O into a set of *SPU* bins. The order level matching fails to jointly consider *SPU* level packing records in different orders and only applicable when historical packing orders that dominate the given order exist. We proceed to propose the *SPU* level matching, which looks into more detailed *SPU* level packing records and is able to regroup them into a packing plan for the given order.

Algorithm 4 Matching by orders ($Matching_{order}()$)

INPUT: a packing order $O = <(t_1, q_1), \ldots, (t_m, q_m)>$, the inverted index T, the cost function $Cost(p)$ and the space wastage ratio threshold ϵ
OUTPUT: the packing plan p for O

```
1: for $i \leftarrow 1$ to $|O|$ do
2:     $record_i \leftarrow set()$
3:     for each $Q_j \geq q_i$ under $item = t_i$ in $T$
4:         $record_i \leftarrow record_i \cup R_j$ //For each item $t_i$, retrieve all *order* level packing records that packs no less than the quantity $q_i$ of $t_i$ via the inverted index $T$
5: $R \leftarrow \cap_{i=1}^{|O|} record_i$
6: $p \leftarrow null$
7: for each order level packing record $r \in R$
8:     if each *SPU* bin's estimated space wastage ratio $\leq \epsilon$
9:         if $p = null$ or $Cost(p) > Cost(r.p)$
10:            $p \leftarrow r.p$
11: return $p$
```

The pseudo code of the *SPU* level matching is given in Algorithm 5. It takes as input a query packing order O, the inverted index T, a function $Cost(p)$ that measures the cost of a packing plan, a space wastage threshold ϵ and a step size τ that used to relax the current space wastage threshold when no feasible *SPU* level packing plan exists. The algorithm works as follows. We first retrieve all *SPU* level packing records R that contain one or more items in O via the inverted index T (Line 1–5). The *SPU* level matching algorithm is inspired by the following two intuitions: one is that we prefer the packing plans with smaller space wastage ratio and the other is that we prefer the packing plans with smaller packing cost. We take a heuristic strategy of repeatedly using an *SPU* bin packing as many unpacked items in the order O as possible according to the historical *SPU* level packing plans, until all items has been packed ($O' = \phi$) or no packing plan that has a space wastage ratio smaller than the threshold ϵ ($O' \neq \phi$ and $\epsilon' > \epsilon$) exists (Line 9–26). As we prefer packing plans with smaller space wastage ratio, we explore the space wastage ratio of the *SPU* bins from small to large, *i.e.,* from 0 to the maximum allowed wastage ratio ϵ, with a step size as τ. For a currently explored space wastage ratio ϵ', we pack as many remained items as each *SPU* level packing record can hold to the corresponding *SPU* bin and we retrieve all the packing plans R' that has a space wastage ratio not greater than ϵ' (Line 11–13). If no such *SPU* level packing plan with a space wastage ratio that does not exceed the current explored ratio ϵ' exists (Line 14–15), we then relax the current ratio by the step size τ (Line 21–25). Otherwise, we select the plan in R' with the minimum cost as the next *SPU* and we remove the items packed in it from the set O' of unpacked items (Line 17–20). The process stops when all items are packed or no *SPU* level packing plan with a space wastage ratio smaller than the threshold exists for the unpacked items. The algorithm then returns a set of *SPU* bins packed with the items in O and the remaining unpacked items if exist (Line 26).

5.5 Matching by SKUs ($Matching_{sku}()$)

After processing the *SPU* level matching, there could still be items that are not packed. We proceed to do the *SKU* level matching to further pack the remaining items by jointly considering *SKU*

Algorithm 5 Matching by SPUs ($Matching_{spu}()$)

INPUT: a query packing order $O =< (t_1, q_1), \ldots, (t_m, q_m) >$, the inverted index T, the cost function $Cost(p)$, the space wastage ratio threshold ϵ and the step size τ

OUTPUT: the packing plan P and the remaining materials O'

```
1:  for $i = 1$ to $|O|$ do
2:      $record_i \leftarrow set()$
3:      for each $Q_j$ under $item = t_i$ in $T$
4:          $record_i \leftarrow record_i \cup R_j$ //For each item $t_i$, retrieve all SPU level packing records that contains $t_i$ via the inverted index $T$
5:  $R \leftarrow \cup_i record_i$
6:  $P \leftarrow \{\}, O' \leftarrow O, \epsilon' \leftarrow 0$
7:  while $O' \neq \phi$ and $\epsilon' \leq \epsilon$ do
8:      $has\_solution \leftarrow True$
9:      while $has\_solution = True$ and $O' \neq \phi$
10:         $R' \leftarrow \{\}$
11:         for each SPU level record $r \in R$
12:             if $\sum_{t \in O' \cap r} t.v \geq (1 - \epsilon') * r.b.v$
13:                 $R' \leftarrow R' \cup (r.b, O' \cap r)$
14:         if $|R'| = 0$
15:             $has\_solution \leftarrow False$
16:         else
17:             find the plan $p$ in $R'$ with the minimum cost
18:             $P \leftarrow P \cup p$
19:             $O' \leftarrow O' - p$
20:             $R' \leftarrow R' \cap O'$
21:         if $has\_solution = False$
22:             $has\_solution \leftarrow True$
23:             $\epsilon' \leftarrow \epsilon' + \tau$
24:             if $\epsilon' > \epsilon$
25:                 break
26: return $P, O'$
```

level records in different *SPU* bins. The processing of the *SKU* level matching is similar to that of the *SPU* level matching. However, it is worth noting that the goal of the MLBP problem is to pack all items into *SPU* bins. The *SKU* bin level matching packs the items into *SKU* bins, which is not the final state. We need to further pack all *SKU* bins into *SPU* bins. This process becomes a traditional bin packing problem as both the items (here refers to *SKU* bins) and the bins (SPU bins) are regular cuboids and their sizes are known. We adopt the algorithm in [13] to do this process.

Due to the lack of historical references, it is possible that there still exist unpacked items after processing the matching algorithm at all levels. We leave these unpacked items to specialized workers for manual packing. It is fortunate that such manual solutions can enrich the record database and is able to guide packing orders in the future. In addition, We remark that the *inverted index* that proposed in Section 4.2.1 to speed up the retrieval of related packing records and the *skyline records* in Section 4.2.2 to reduce the packing record database are still applicable in the matching algorithms.

6 EMPIRICAL STUDY

6.1 Experimental Setup

6.1.1 Algorithms for Comparison. We consider the dynamic programming algorithm developed in Section 4 (***DP***) and the multi-level matching algorithm in Section 5. The matching algorithm are considered in different levels, *i.e.*, singly *order* level matching (***Order Level***), both *order* and *SPU* level matching (***SPU Level***), and *order+SPU+SKU* level matching (***SKU Level***). We consider the cost function as minimizing the number of used bins. Unless specified otherwise, all algorithms are implemented with optimizations.

6.1.2 System Setup and Metrics. The experiments are conducted on a Spark cluster with 50 cores. Each core is equipped with an Intel(R) Xeon(R) CPU E7-4820 v2 @ 2.00GHz. The driver memory is 10G and the executor memory are 20G. The algorithms are implemented in Python. The inverted index introduced in Section 4.2.1 is in-memory.

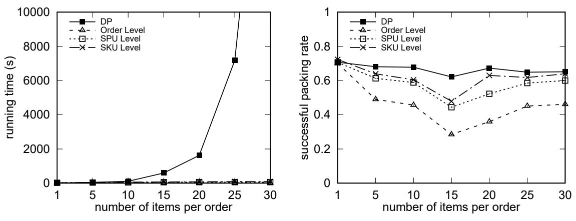
To simulate the real case, the query packing orders are processed in batches. Each batch contains 5, 000 packing orders. We measure the running time of processing one batch of orders and the successful packing rate, *i.e.*, the proportion of orders in the batch that the algorithm obtains applicable packing plans. For each experiment, we process 200 batches of orders and report the average result.

6.1.3 Dataset. We test the algorithms using the real packing dataset in Huawei Process & Engineering System. Packing records from January to October 2017 are treated as the historical database. It contains 58, 511 different items and 1, 715 different bins, including 1, 662 types of *SPU* bins and 53 kinds of *SKU* bins, respectively. There are 4, 951, 436 order level packing records, which can be further decomposed into 8, 377, 100 *SPU* level records.

6.2 Empirical Results

6.2.1 Dynamic Programming versus Fuzzy Matching.

A. Varying the total number of items in a packing order. In this set of experiment, we use the packing records in October 2017 as the historical database. We generate 7 batches of query packing orders. Each batch contains 5, 000 packing orders that are randomly picked with replacement from the packing orders in November 2017, with the requirement that orders in the same batch ask to pack the same total number of items. For example, all orders in the first batch ask to pack only one single item; and each order in the second batch is to pack 5 items. The result are shown in Fig. 4. We can observe that, when the number of items increases, the running time of DP nearly exponentially increases while that of matching algorithms are almost unaffected. Recall that the total number of the states in DP for a packing order O is $\prod_{i=1}^{|O|}(q_i + 1)$. With the increasing of $\sum_{i=1}^{|O|} q_i$, the number of states in DP has higher chances to rapidly grow. That explains the trend of the running time of the DP algorithm. As for the successful packing rate, a general observation is that *Order Level < SPU Level < SKU Level < DP.*



(a) Running Time (b) Successful Packing Rate

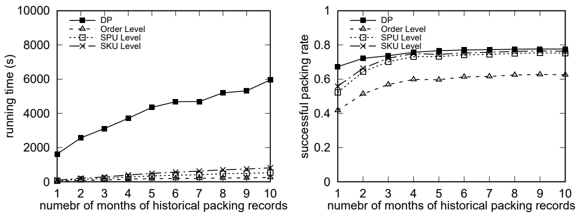
Figure 4: Varying number of items (DP vs. fuzzy matching)

B. Varying size of historical packing record database. In this set of experiments, the query packing orders in each batch are randomly selected from packing orders containing 20 items in November 2017. We vary the packing record database from one month's to ten months' data before October 2017. That is, one month's data refers to using packing records in October 2017 as the historical database; two months' data refers to using records in September and October 2017; and so on. Table 1 illustrates the sizes of the databases and their inverted indexes, as well as the off-line construction times of the indexes. As we can see, with the increase of the database size, the inverted index size grows linearly, while the off-line construction time of the index increases sub-linearly.

Table 1: Inverted Index Experimental Results

Database Size (MB)	Index Size (MB)	Construction Time (s)
618	71	61
1267	146	64
2126	216	61
2980	293	63
3897	382	66
4835	462	74
5773	538	76
6576	611	89
7386	667	97
8152	722	97

The experimental results are illustrated in Fig. 5. The running time of all algorithms increases when the database is enlarged, but that of the matching algorithms grows much slower. It is also observable that the enlargement of historical data improves the successful packing rate of all algorithms, but the matching algorithms benefit more. For example, when enlarging the database from one month to ten months, the successful packing rate of *SPU* level matching improves from 52.24% to 75.18%, while that of the DP algorithm only increases from 67.28% to 77.64%.



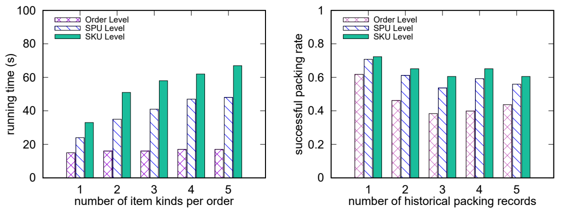
(a) Running Time (b) Successful Packing Rate

Figure 5: Varying size of database (DP vs. fuzzy matching)

From the above two sets of experiments, we observe that although the DP algorithm achieves better solutions, it is inapplicable for packing orders with large sizes. The matching algorithms are more efficient and scale better than DP either when the packing order size or the historical database size increases. The DP algorithm generally achieves a higher successful packing rate than the fuzzy matching. But the *SPU* and *SKU* level matching algorithms have the trend to achieve a successful packing rate very close to that of DP, if enough historical data is available.

In the following experiments, we focus on examining the performance of the fuzzy matching algorithm in different levels.

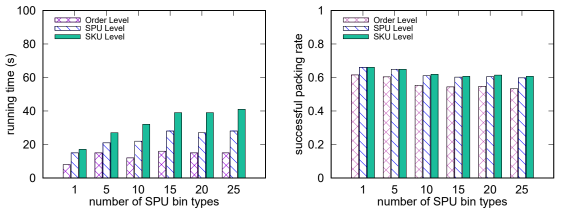
6.2.2 Varying the number of item kinds in the packing order. In this set of experiments, we use the packing records in October 2017 as the historical database. Five batches of query packing orders are randomly generated from orders in November 2017, where the orders in the same batch contain the same number of item kinds. More specifically, all orders in the first batch ask to pack one single kind of items and the second batch is consisted of orders that are to pack two kinds of items. The results are shown in Fig. 6. Generally, the running time of the algorithms increases along with the increasing of the item kinds in the order. The reason is that, more possible combinations of items exist when the number of item kinds increases. The successful packing rates have a slight trend to decrease when the number of item kinds increases. Recall that the idea in matching algorithms is based on the fact that similar packing orders exist in the history. As there are more different combinations of items in packing orders, when the number of item kinds increases. It reduces the chance to find an identical or similar packing record in the history and thus explains this trend.



(a) Running Time (b) Successful Packing Rate

Figure 6: Varying # item kinds in the packing order

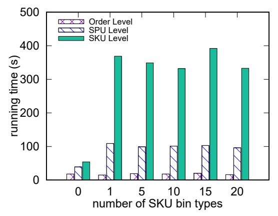
6.2.3 Varying number of SPU bin types. The total number of bin types are different in packing scenarios of different enterprises. We here evaluate how the number of *SPU* bin types affects the performance of the algorithms. For each experiment, we generate a set of *SPU* bin types. The cardinalities of these sets vary from 1, 5, 10, 15, 20 to 25. We consider the historical database as the packing records in October 2017 that use only *SPU* bins in the corresponding bin set. The query packing orders are randomly selected from orders in November that contain only items in the corresponding database. As illustrated in Fig. 7, when the number of *SPU* bin types increases, the running time of the matching algorithms rises and the successful packing rate is almost unaffected.



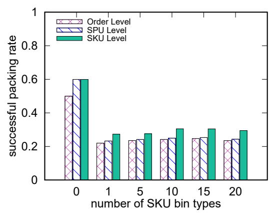
(a) Running Time (b) Successful Packing Rate

Figure 7: Varying SPU bin types

6.2.4 Varying number of SKU bin types. We proceed to study the effects of the number of available *SKU* bin types. Similarly, we select 6 sets of *SKU* bins, where each set respectively contains 0, 1, 5, 10, 15 and 20 different *SKU* bins. For each experiment, The historical database is considered as the packing records in October that only contain *SKU* bins in the corresponding *SKU* bin set. The query orders are selected from orders in November that contain only items in the corresponding historical database. The results is given in Fig. 8. As we can see, the running time of *SKU Level* matching increase dramatically when number of *SKU* bin types > 0. The



(a) Running Time



(b) Successful Packing Rate

Figure 8: Varying SKU bin types

reason is that, unlike the real cases that a great proportion of packing records do not contain *SKU* bins, here *SKU* bins exist in each historical record in the database. It results in a larger search space for the *SKU* level matching. Another observation is that successful packing rate decreases to around 30% when orders are restricted to contain *SKU* bins. It is because packing orders that uses *SKU* bins usually contain more items and have more complex packing structures than those only containing *SPU* bins. It is therefore less likely to find a similar match in the history.

6.2.5 On-line performance vs. manual packing. At last, we present the real on-line performance of the proposed matching algorithms. We deployed the *SPU level* matching algorithm in Huawei Process & Engineering System. The *SPU* level matching is selected based on the following observations. First, the *SPU level* matching is more efficient and scales better than DP; and it achieves a successful packing rate very close to that of DP when enough historical packing records is available. Second, comparing to singly using *Order level* matching, *SPU level* matching greatly improves the successful packing rate by 12.74% on average. Third, the *SKU level* matching only improves the successful packing rate by an average of 1.6% while taking averagely 1.7 times of execution time comparing to the *SPU level* matching. Moreover, the usage rate of *SKU* bins is small in the packing scenario in Huawei. Each packing record in the history contains only averagely 0.21 different *SKU* bins.

The on-line *SPU level* matching algorithm considers the packing records in previous 6 months as historical record database. It takes approximately 8 minutes to process a batching of 5,000 query packing orders. The on-line successful packing rate is 80.54% on average. It releases more than 30% workloads. As the algorithm is to "*copy*" the historical packing plans for future orders, we see no obvious reduction on the number/cost of used bins. However, it tends to make the packing plans for similar orders be consistent. It therefore internally makes the packing line be easier to operate/manage and externally improves the customer satisfaction.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we consider a 3D real-life MLBP problem, where no explicit objective function exists, and the item sizes and the packing constraints are unknown. We solve the problem in a data-driven perspective by using manual historical packing records. We propose both a dynamic programming approach to find optimal solutions for problems with normal sizes and a multi-level fuzzy matching algorithm to deal with problems of large sizes while guarantees the maximum space wastage ratio. Several optimization techniques have been proposed to accelerate the processing. Extensive experiments on real datasets demonstrate that the proposed algorithms scale well and achieve good performance under a wide range of system settings. Moreover, we have deployed the algorithms in Huawei Process & Engineering System, which achieves a performance of accelerating the execution time of processing 5,000 packing orders to about 8 minutes with an average successful packing rate as 80.54%, which releases at least 30% workloads of packing workers.

ACKNOWLEDGMENTS

Special thanks to Riemann Data Studio and Logistics Department of Huawei Supply Chain. The work is also partially supported by the Hong Kong RGC CRF C6030-18G, GRF Project 16207617, and Hong Kong ITC ITF grants ITS/391/15FX and ITS/212/16FP.

REFERENCES

[1] B B. H. Korte and Jens Vygen. 2012. *Combinatorial Optimization: Theory and Algorithms*. Vol. 21.
[2] Christian Blum and Verena Schmid. 2013. Solving the 2D Bin Packing Problem by Means of a Hybrid Evolutionary Algorithm. *Procedia Computer Science* 18 (2013), 899–908.
[3] C.S. Chen, S.M. Lee, and Q.S. Shen. 1995. An analytical model for the container loading problem. *European Journal of Operational Research* 80, 1 (1995), 68 – 76.
[4] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. 2017. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review* 24 (2017), 63 – 79.
[5] Edward G. Coffman Jr., János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. 2013. *Bin Packing Approximation Algorithms: Survey and Classification*. Springer New York, New York, NY, 455–531.
[6] D. Friesen and M. Langston. 1986. Variable Sized Bin Packing. *SIAM J. Comput.* 15, 1 (1986), 222–230.
[7] J.A. George and D.F. Robinson. 1980. A heuristic for packing boxes into a container. *Computers & Operations Research* 7, 3 (1980), 147 – 156.
[8] José Fernando Gonçalves and Mauricio G.C. Resende. 2013. A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics* 145, 2 (2013), 500 – 510.
[9] Mohamed Haouari and Mehdi Serairi. 2009. Heuristics for the variable sized bin-packing problem. *Computers & Operations Research* 36, 10 (2009), 2877 – 2884.
[10] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. 2017. Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method. *CoRR* abs/1708.05930 (2017).
[11] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. 1974. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM J. Comput.* 3, 4 (1974), 299–325.
[12] Kyungdaw Kang, Ilkyeong Moon, and Hongfeng Wang. 2012. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Appl. Math. Comput.* 219, 3 (2012), 1287 – 1299.
[13] Xijun Li, Mingxuan Yuan, Di Chen, Jianguo Yao, and Jia Zeng. 2018. A Data-Driven Three-Layer Algorithm for Split Delivery Vehicle Routing Problem with 3D Container Loading Constraint. (2018), 528–536.
[14] Andrea Lodi, Silvano Martello, and Daniele Vigo. 2002. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research* 141, 2 (2002), 410 – 420.
[15] Eunice López-Camacho, Hugo Terashima-Marín, Gabriela Ochoa, and Santiago Enrique Conant-Pablos. 2013. Understanding the structure of bin packing problems through principal component analysis. *International Journal of Production Economics* 145, 2 (2013), 488 – 499.
[16] Mohamed Maiza, Abdenour Labed, and Mohammed Said Radjef. 2013. Efficient algorithms for the offline variable sized bin-packing problem. *Journal of Global Optimization* 57, 3 (2013), 1025–1038.
[17] Feng Mao, Edgar Blanco, Mingang Fu, Rohit Jain, Anurag Gupta, Sebastien Mancel, Rong Yuan, Stephen Guo, Sai Kumar, and Yayang Tian. 2017. Small Boxes Big Data: A Deep Learning Approach to Optimize Variable Sized Bin Packing. *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)* (2017), 80–89.
[18] Silvano Martello, David Pisinger, and Daniele Vigo. 1998. The Three-Dimensional Bin Packing Problem. *Operations Research* 48 (1998).
[19] Guido Perboli, Luca Gobbato, and Francesca Perfetti. 2014. Packing Problems in Transportation and Supply Chain: New Problems and Trends. *Procedia - Social and Behavioral Sciences* 111 (2014), 672 – 681.