Omega 50 (2015) 43–53

Contents lists available at ScienceDirect

Omega

journal homepage: www.elsevier.com/locate/omega

# The two-dimensional vector packing problem with piecewise linear cost function ☆

Qian Hu [a], Andrew Lim [a], Wenbin Zhu [b,*]

[a] International Center of Management Science and Engineering, School of Management and Engineering, Nanjing University, Nanjing 210093, China
[b] School of Business Administration, South China University of Technology, Guangzhou 510640, China

ARTICLE INFO

*Article history:*
Received 26 November 2012
Accepted 12 July 2014
This paper was processed by Associate Editor Romeijn.
Available online 19 July 2014

*Keywords:*
Heuristic
Application
Bin packing
Two-dimensional vector packing
Piecewise linear cost function

ABSTRACT

The two-dimensional vector packing problem with piecewise linear cost function (2DVPP-PLC) is a practical problem faced by a manufacturer of children's apparel that ships products using courier service. The manufacturer must ship a number of items using standard-sized cartons, where the cost of a carton quoted by the courier is determined by a piecewise linear function of its weight. The cost function is not necessarily convex or concave. The objective is to pack all given items into a set of cartons such that the total delivery cost is minimized while observing both the weight limit and volume capacity constraints. This problem is commonly faced by many manufacturers that ship products using courier service. We formulate the problem as an integer programming model. Since the 2DVPP-PLC generalizes the classical bin packing problem, it is more complex and challenging. Solving it directly using CPLEX is successful only for small instances. We propose a simple heuristic that is extremely fast and produces high-quality solutions for instances of practical size. We develop an iterative local search algorithm to improve the solution quality further. We generate two categories of test data that can serve as benchmark for future research.



## 1. Introduction

We consider a new extension to the bin packing problem, the two-dimensional vector packing problem with piecewise linear cost function (2DVPP-PLC). In the 2DVPP-PLC, a set of items are to be packed into identical bins so that the total cost of utilized bins is minimized. Each item has two attributes: the weight and the volume. A set of items can be packed into a bin if their total weight and volume do not exceed the capacities of the bin. The cost of a bin is a piecewise linear function of the total weight of packed items.

This problem is motivated by a project awarded by a manufacturer of children's apparel with several production bases and hundreds of retail stores located across the globe. The manufacturer distributes its products from its production bases to retail stores through an express courier company such as Federal Express or DHL under a long-term contract. Articles of children's apparel (such as shorts, jackets and rompers) of the same style and size are bundled to form items; for example, one item may be a bundle of two dozen rompers. In each delivery, a batch of items produced by a production base are packed into cartons and delivered to a specific store by a courier company. To simplify the operations of packing, storage and transportation, the manufacturer employs only one type of carton with fixed dimensions in each delivery.

The delivery cost for a carton is calculated based on the weight of the carton, the source and the destination. The destinations (stores) are grouped into zones. For a given production base and the destination zone, the delivery cost for a carton is a function of the total weight of the items packed into that carton (we assume that the net weight of the carton is negligible). We obtained from the manufacturer the pricing function offered by the courier company to deliver one carton from a certain production base to each of the seven zones, as shown in Fig. 1(a), where the $x$-axis represents the total weight of the items in pounds (lb) in a carton and the $y$-axis corresponds to the delivery cost of that carton in US dollars. We approximate the pricing function for each zone with a piecewise linear cost function. For example, the pricing function for Zone 3 is approximated by the following function (Fig. 1(b)):

$$f(x)=\begin{cases}0 & \text{if } x=0;\\ 5 & \text{if } x\in(0,10];\\ 0.2x+3 & \text{if } x\in(10,70];\\ 0.5x-18 & \text{if } x\in(70,150].\end{cases} \quad (1)$$

We have conferred with the courier company on the rationale behind this pricing scheme, which was devised based on practical

☆This manuscript was processed by Associate Editor Romeijn.

* Corresponding author. Tel.: +852 64067667.
*E-mail addresses:* janoy.qian@gmail.com (Q. Hu), alim.china@gmail.com (A. Lim), i@zhuwb.com (W. Zhu).

http://dx.doi.org/10.1016/j.omega.2014.07.004
0305-0483/

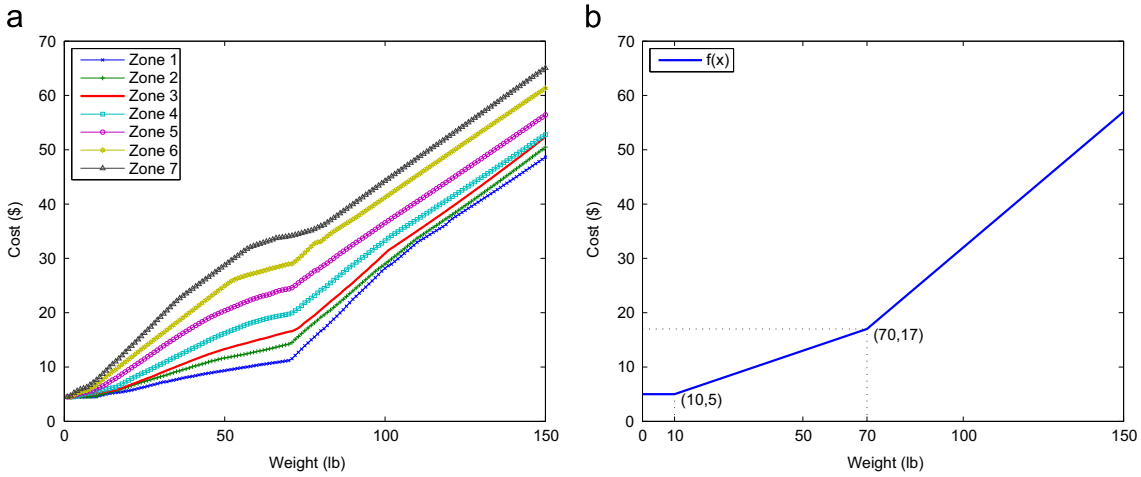


**Fig. 1.** (a) Pricing schemes for seven zones. (b) Piecewise linear function for Zone 3.

considerations. Delivering each carton incurs a fixed transaction cost, so the courier company charges US$5 for cartons with weight not more than 10 lb. When the carton weight exceeds 10 lb, the delivery cost increases with weight at a fixed rate of 20 cents per lb. However, this rate increases to 50 cents per lb when the carton weight is greater than 70 lb. One of the reasons is that light cartons can be handled manually by a single worker with ease, while overweight cartons require special handling equipment or more than one worker, which increases the operation cost considerably. Moreover, the carton weight is capped by a specific limit (150 lb in our case).

Since textile packages are flexible and non-fragile, the manufacturer only needs to consider the weight and the volume of each item during the packing process, i.e., we can safely assume that a set of items can fit into the carton as long as the total volume and weight of the items do not exceed the volume capacity and the weight limit of the carton, respectively. Therefore, minimizing shipping cost of a delivery can be modeled as the two-dimensional vector packing problem with piecewise linear cost function (2DVPP-PLC).

Both the two-dimensional vector packing problems (and many related variants of bin packing problems) and the piecewise linear cost function (in the context of optimization problems) have received considerable attention (Section 2). However, there is no prior literature dedicated to the 2DVPP-PLC, to the best of our knowledge, despite its value to practitioners. We, therefore, initiate the investigation of this problem by formally introducing the problem and presenting an integer programming (IP) formulation (Section 3). The 2DVPP-PLC as a generalization of bin packing problem is clearly NP hard. Solving the IP formulation using standard IP solver, such as CPLEX, is viable only for small problem instances. We resort to heuristics to produce practical solutions for large instances for our client. We adapt the classical first-fit and best-fit heuristics to generate initial solutions (Section 4). Although first-fit and best-fit produce good initial solutions, both are incomplete in the sense that all optimal solutions may be missed for some problem instances. We extend a shortest-path based heuristic from Haouari and Serairi [14] to the 2DVPP-PLC, which is complete (Section 5). Based on this shortest-path heuristic, we develop an iterative local search algorithm to improve the solution quality (Section 6). We generate two categories of test instances: **opt**, where an optimal solution is known; and **rand**, where no optimal solution is known. Computational experiments on **opt** instances show that our proposed algorithm is capable of finding high-quality solutions in a reasonable computation time. The category **rand** includes a comprehensive set of instances of various scales and can serve as benchmark for future research.

## 2. Literature review

Packing and cutting are important processes in production and logistics. There is a large number of publications in the area of packing and cutting [8,25,2] over the decades. Wscher et al. [30] provided a typology to organize and categorize existing literature.

The 2DVPP-PLC is one of many generalizations of the classical one-dimensional bin packing problem (BPP) [9,12]. The rich body of literature on many variants of the BPP includes other generalizations such as the two-dimensional (rectangle) bin packing problem (2D-BPP) [22,23], which extends both bins and items to be rectangles and requires that the items must be orthogonally packed into the bins without overlapping; the three-dimensional (rectangle) bin packing problem (3D-BPP) [21,20,32,31,5], which attempts to orthogonally pack boxes into containers without overlapping; the bin packing problem with conflicts (BPP-conflicts) [13,24], which specifies that some pairs of items cannot be loaded into the same bin; the two-dimensional vector packing problem (2DVPP) [26,3], which adds a second attribute (in addition to volume) and requires that the total for both attributes in a single bin must be within the specified limits; and the variable-sized bin packing problem (VSBPP) [17,14,15,28], which considers different sizes of bins each with a different specified cost; the bin packing problem with general cost structure (BPP-GC) [1,10], which introduces a general cost structure when calculating the cost of a bin.

The 2DVPP-PLC generalizes the 2DVPP. While the 2DVPP aims to minimize the number of bins used, the 2DVPP-PLC extends and targets to minimize the total cost which depends on the given piecewise linear cost function. In the 2DVPP-PLC, minimizing cost does not imply that the number of bins used is minimized. In fact, splitting an overly filled bin into two bins may reduce the total shipping cost (we provide an illustrative example of the difference between the 2DVPP and the 2DVPP-PLC in Appendix A in the online supplements). In the existing literature, the 2DVPP has been solved mainly using two classes of algorithms: exact algorithms and heuristics. Woeginger [29] proved that there is no polynomial-time approximation algorithm with a performance guarantee of less than or equal to $1+\epsilon$ (where $\epsilon$ tends to zero) for the 2DVPP. Subsequently, Kellerer and Kotov [18] designed an $O(n \log n)$ approximation algorithm that has an absolute worst-case

performance ratio of 2. To solve the 2DVPP optimally, Spieksma [26] proposed a branch-and-bound algorithm, and Caprara and Toth [3] proposed three branch-and-bound and one branch-and-price algorithms. Caprara and Toth [3] also introduced several heuristics, including several adaptations of greedy heuristics for the classical BPP, a constructive heuristic that iteratively solves the maximum-profit matching problem based on a compatibility graph, and a linear programming rounding heuristic.

For the bin packing problems with general cost structure, the objective becomes minimizing the total cost rather than minimizing the number of bins used. There are two predominant types of general cost structure investigated in the current literature: one is to define the cost of a bin as a concave and monotone function of the number of items packed into the bin [1,10], and the other is to define the cost of a bin as a non-decreasing concave function of the utilization of the bin [19]. These analyses of average-case or worst-case performance of proposed heuristics or approximation algorithms rely on the concavity of the cost function. In general, the cost function of a practical 2DVPP-PLC problem instance is neither convex nor concave (see Fig. 1(b)). Therefore, it is unlikely that such heuristics or approximation algorithm can be directly extended to the 2DVPP-PLC with similar analytical results.

In many application domains, such as transportation and production planning, the cost function is nonlinear due to factors such as economics of scale, fixed charges and discounts for high volume. Such nonlinear cost functions are often approximated by piecewise-linear functions (usually nonconvex). Optimization problems with piecewise linear cost function can be modeled as mixed integer programming (MIP) by introducing integer variables to model the costs. Croxton et al. [6] have shown that when the cost function is separable, the three textbook models—the so-called incremental, multiple choice and convex combination models—have equivalent linear programming relaxations. When the cost function is nonseparable, Vielma et al. [27] compared the theoretical and computational aspects of various MIP models (including the three textbook models). In particular, the multiple choice model is shown to be one of the best when the number of polytopes defining the piecewise linear function is small. For specific optimization problems with piecewise linear cost function, specialized MIP models and algorithms have been investigated. For example, knapsack problems [16], procurement auctions [11], supply chains [4,7].

## 3. Problem definition and formulation

In the 2DVPP-PLC, we are given $n$ items, $I=\{1,2,\ldots,n\}$, each with volume $v_i$ and weight $w_i$ and unlimited supply of identical bins with volume capacity $V$ and weight limit $W$. The cost of a bin is a piecewise linear function $f(x)$ of the total weight $x$ of the items packed into that bin. The cost function is non-decreasing and non-negative on $[0,W]$, however, it need not to be convex or concave. We want to devise a packing plan that places all items into bins such that the total cost of all used bins is minimized. Without loss of generality, we assume every item is small enough to be packed into a bin, so that a feasible solution always exists. All problem input $w_i$, $v_i$, $W$ and $V$ are integers.

Let $B$ be the set of bins available. The $k$th piece of cost function is given by $f(x)=s_k x+b_k$ for $x\in(e_{k-1},e_k]$, where $e_{k-1}<e_k$ for all $k\in K=\{1,2,\ldots\}$. We set a binary variable $y_j^k$ to 1 the cost of bin $j$ should be evaluated by the $k$th piece of the cost function, and 0 otherwise. We set a binary variable $x_{i,j}^k$ to 1 if item $i$ is packed into bin $j$ where the cost of the bin is evaluated by the $k$th piece of the cost function. We can formulate the 2DVPP-PLC using the following multiple choice model:

$$\text{Minimize} \quad \sum_{j\in B}\sum_{k\in K}\left(s_k\sum_{i\in I}w_i x_{i,j}^k+b_k y_j^k\right) \tag{2}$$

$$\text{Subject to} \quad \sum_{j\in B}\sum_{k\in K}x_{i,j}^k\geq 1, \quad \forall i\in I \tag{3}$$

$$(e_{k-1}+1)y_j^k\leq\sum_{i\in I}w_i x_{i,j}^k\leq e_k y_j^k, \quad \forall j\in B,\ k\in K \tag{4}$$

$$\sum_{k\in K}y_j^k\leq 1, \quad \forall j\in B \tag{5}$$

$$\sum_{i\in I}\sum_{k\in K}v_i x_{i,j}^k\leq V, \quad \forall j\in B \tag{6}$$

$$x_{i,j}^k\in\{0,1\}, \quad \forall i\in I,\ j\in B,\ k\in K \tag{7}$$

$$y_j^k\in\{0,1\}, \quad \forall j\in B,\ k\in K \tag{8}$$

Constraints (4) stipulate that the cost of a bin should be evaluated by the $k$th piece of the cost function if and only if the total weight of the items in the bin falls in $(e_{k-1},e_k]$. Since all item weights are integers, we can replace $(e_{k-1},e_k]$ by $[e_{k-1}+1,e_k]$. Constraints (5) enforce that at most one piece of the cost function is used to evaluate the cost of a bin. Constraints (3) require all items that are packed into some bin. Constraints (6) ensure that the total volume of items in a bin does not exceed the volume capacity of the bin. The weight limit of a bin is enforced by constraints (4) by setting $e_{|K|}=W$.

For any bin $j$, there are two cases. When there are items in it, constraints (4) and (5) will ensure exactly one piece of the cost function evaluates its cost. When there is no item in it, constraints (4) will ensure that $y_j^k=0$ for all $k$. In either case, the cost of bin $j$ is given by

$$\sum_{k\in K}\left(s_k\sum_{i\in I}w_i x_{i,j}^k+b_k y_j^k\right)$$

Hence objective (2) minimizes total cost of all bins.

When the number of bins $|B|$ is sufficiently large, our model will guarantee an optimal solution to the 2DVPP-PLC, a trivial choice of $|B|$ is the total number of items. From efficiency point of view, we would like to set $|B|$ as small as possible, since the complexity of our model (in terms of the number of constraints and decision variables) is directly proportional to $|B|$. Determining the optimal $|B|$ remains an open research question.

Although our formulation is a correct model for the 2DVPP-PLC, solving it directly by a standard solver such as ILOG CPLEX is impractical. Such an approach is viable only for small instances according to our experiments (Section 7). It is therefore necessary to investigate heuristic approaches that are more efficient for large instances.

## 4. Greedy-on-weight-range heuristic

In this section, we first extend the well-known first-fit and best-fit heuristics for the classical bin packing problem to the 2DVPP-PLC. We then proceed to develop the Greedy-on-Weight-Range heuristic, which makes use of first-fit and best-fit to produce feasible solutions to the 2DVPP-PLC.

Both first-fit and best-fit heuristics load items one by one according to a loading sequence. The first-fit (resp. best-fit) heuristic places an item into the first (resp. best) bin that can accommodate the item. If no existing bin can accommodate an item, a new bin is employed to pack the item. In the best-fit heuristic, the best bin is the bin with the smallest remaining weight or volume in percentage term before the item is packed. More precisely, let $\mathrm{rw}(b,i)$ and $\mathrm{rv}(b,i)$ be the remaining weight and

remaining volume in bin $b$ immediately before item $i$ is packed, respectively. The best bin to pack item $i$ is the bin that minimizes the following measure:

$$\text{eval}(b,i) = \min\left\{\frac{\text{rw}(b,i)}{W}, \frac{\text{rv}(b,i)}{V}\right\}$$

If there is a tie, we choose the earliest such bin.

Both the first-fit and the best-fit attempt to fill each bin as full as possible. Such greedy rule is aligned with the objective of the bin packing problem—minimizing the total number of bins. However, it is not aligned with the objective of the 2DVPP-PLC. For example, when the cost function shown in Fig. 1(b) is used, packing two identical items with both weight and volume 70 into a single bin with both weight limit and volume capacity 150 will cost $f(140)=52$. In contrast, packing the two items into separate bins will only cost $f(70)+f(70)=34$. In fact, since the total weight of all items is fixed, minimizing total cost is equivalent to minimizing average cost per unit weight. Therefore, minimizing the average cost per unit weight for each bin is likely to reduce the total cost. To implement this idea, we impose an artificial weight limit $l \le W$ for each bin when using first-fit or best-fit. The natural choice of $l$ is the value(s) that minimizes $g(l)=f(l)/l$, where $g(l)$ is the average cost per unit weight for a bin with weight $l$.

When we impose an artificial weight limit $l^*$ that minimizes $g(l)$, the weight of the items in some bins may be less than $l^*$. As a result the effective average cost per unit weight in the solution will slightly deviate from $g(l^*)$. It is possible that using a slightly larger $l$ will result in a solution with smaller total cost than $l^*$. We consider a few values of $l$ in the non-decreasing order of $g(l)$ and take the best solution. We call our approach greedy-on-weight-range GWR, see Algorithm 1. The GWR procedure takes four inputs. The first is a set of items $I$ to be loaded. The second is a sorting rule $SR$ that will sort all items into a loading sequence. The third is a decoder $DR$ such as first-fit or best-fit heuristic that will convert a loading sequence into a solution. The last parameter $L$ specifies the number of artificial weight limits we will consider.

**Algorithm 1.** Greedy-on-weight-range heuristic.

```
GWR (I, SR, DR, L)
    // Inputs
    // I: the set of items to be loaded
    // SR: a sorting rule
    // DR: a decoder such as first-fit or best-fit
    // L: the number of artificial weight limits
1   bestSol =NULL;
2   Let {l_1, l_2, ..., l_L} be artificial weight limits in ascending order
    of g(l);
3   π = sort items I according to sorting rule SR;
4   for each l ∈ {l_1, l_2, ..., l_L}
5     π' = π;
6     B = ∅;
7     for each item i ∈ π with weight exceeding l
8       allocate a bin b for the item i;
9       append b to the end of B;
10      delete i from π';
11    Convert sequence π' using decoder DR into a solution B';
12    Append bins in B' to the end of B;
13    Update the bestSol, if B represents a better solution;
14  return bestSol
```

We consider all possible rules for sorting the items by weight and volume as listed below:

- Weight ascending, then volume ascending (WAVA)
- Weight ascending, then volume descending (WAVD)
- Weight descending, then volume ascending (WDVA)
- Weight descending, then volume descending (WDVD)
- Volume ascending, then weight ascending (VAWA)
- Volume ascending, then weight descending (VAWD)
- Volume descending, then weight ascending (VDWA)
- Volume descending, then weight descending (VDWD)

The two descending sorting rules WDVD and VDWD are expected to perform better than the others, since intuitively treating heavier items first leaves more flexibility to the greedy heuristic as it approaches the end of the construction of a complete solution. We performed a set of computational experiments to evaluate the effectiveness of each sorting rule in the GWR heuristic, and found that WDVD and VDWD are indeed superior (see Section 7.2).

## 5. Sequence based model and shortest-path decoder

Meta-heuristics for the bin packing problem often employ a sequence based model: a solution is encoded as a sequence (permutation) of items. The search is carried out in the space of all possible permutations of items. Each sequence is converted into a solution by a decoder such as first-fit and best-fit heuristics. Among the exponential number of sequences, if there always exists at least one permutation of items which can be converted into an optimal solution by a heuristic, we say this heuristic is *complete*. The sequence based model using either first-fit or best-fit as a decoder is complete for the bin packing problem.

Completeness allows us to reduce the bin packing problem to the searching for a permutation of items. Such reduction effectively reduces the search space by a factor of $\binom{n+m-1}{n-1}$, where $n$ is the total number of items and $m$ is the total number of bins. This is because, for each permutation of $n$ items, there are $\binom{n+m-1}{n-1}$ different ways to arrange them into $m$ or less bins, each arrangement may correspond to a feasible solution. Without completeness, we have to try all possible arrangements to find an optimal solution in the worst case. With completeness property, it is sufficient to convert each permutation of items into one solution.

The completeness of first-fit and best-fit heuristics is lost when they are extended to the 2DVPP-PLC. There are 2DVPP-PLC instances such that none of the $n!$ permutations of items can be converted into any optimal solution. We can construct an example as follows. Given two items of the same type with volume $v=70$ and weight $w=70$, enough bins with capacities $V=150$ and $W=150$, and the cost function shown in Fig. 1(b). An optimal solution would use two bins for a cost of $f(70)+f(70)=34$, but both first-fit and best-fit will produce a solution that uses only one bin for any input permutation with a cost of $f(140)=52$.

We proceed to develop a shortest-path (SP) decoder for 2DVPP-PLC, which restores the desired completeness property. Our shortest-path decoder is based on the idea of Haouari and Serairi [14] for the variable bin-size bin packing problem. Given a sequence of items, we try to partition the sequence into non-overlapping segments such that each segment is a set of items in a bin. The optimal partition of the sequence corresponds to a shortest path in a graph associated with the input sequence.

Given an input sequence $\pi$, we first construct a weighted acyclic graph $G=(V,E)$. The vertex set $V=\{1,\ldots,n,n+1\}$, where $n+1$ is a dummy vertex. The directed edge set $E$ is constructed as follows: (1) add an edge $(i,i+1)$ for $i=1,2,\ldots,n$, the cost of which is $f(w_{\pi(i)})$; (2) add an edge $(i,j)$ for $1\le i<j\le n+1$ if $\sum_{k=i}^{j-1} v_{\pi(k)} \le V$ and $\sum_{k=i}^{j-1} w_{\pi(k)} \le W$, the cost of which is $f(\sum_{k=i}^{j-1} w_{\pi(k)})$ as calculated by the given piecewise linear cost function. We call $G$ the *cost graph* of $\pi$.

It is apparent that any path in $G$ from vertex 1 to vertex $n+1$ corresponds to a feasible 2DVPP-PLC solution, where each edge $(i,j)$ in the path indicates that all items $\pi(i), \pi(i+1), \ldots, \pi(j-1)$ are in the same bin, and the cost of the solution is the total cost of all edges in the path. Furthermore, since every possible partition of the input sequence is captured by a path in graph $G$, the best feasible partition of the sequence can be found by solving a shortest path problem from vertex 1 to $n+1$ on $G$ in $O(|E|)$ time.

To illustrate the SP decoder, we use the example problem instance with $W=V=150$ and item information shown in Table 1; the cost function for each bin is given by Fig. 1(b). Fig. 2 shows the shortest-path decoding process when decoding the sequence $\pi=(3,2,7,10,4,6,9,1,5,8)$, where notations $W_i$, $V_i$ and $C_i$ denote the actual weight, the used volume and the cost of the bin, respectively.

The shortest-path decoder is complete (the proof is given in Appendix B in the online supplements).

**Proposition 1.** *For any instance of the 2DVPP-PLC, there exists an optimal solution that corresponds to the shortest-path decoding of some permutation of the items.*

## 6. An iterative local search algorithm

Since shortest-path decoder is complete, we can search in the space of all permutations of input items to find an optimal solution to the 2DVPP-PLC. We design an iterative local search (ILS) algorithm to explore the permutations of input items. Our ILS algorithm makes use of three neighborhood operators: BIN-SHUFFLE, BIN-SHAKE and SHUFFLE-SHAKE. Given a sequence (permutation) of items, all three operators will transform it into a new sequence and the solution derived from the new sequence using the shortest-path decoder will never be worse than that from the original sequence.

**Table 1**
An problem instance.

| Item ($i$) | Weight ($w_i$) | Volume ($v_i$) | Item ($i$) | Weight ($w_i$) | Volume ($v_i$) |
|---|---|---|---|---|---|
| 1 | 30 | 10 | 6 | 30 | 60 |
| 2 | 30 | 20 | 7 | 20 | 20 |
| 3 | 30 | 30 | 8 | 20 | 30 |
| 4 | 30 | 40 | 9 | 20 | 40 |
| 5 | 30 | 50 | 10 | 20 | 50 |

The first operator BIN-SHUFFLE changes the relative position of some bins while preserving the relative position of the items inside each bin. Let $\mathrm{SP}(\pi)=(s_1,s_2,\ldots,s_k)$ be the solution produced by the SP decoder on sequence $\pi$, where each element $s_i$ is a segment of $\pi$ corresponding to all items in a single bin. We can produce a solution $S'$ from $\mathrm{SP}(\pi)$ by randomly permuting the segments of $\mathrm{SP}(\pi)$, while the order of the items in each segment is unchanged. Clearly the cost of $S'$ is the same as that of $\mathrm{SP}(\pi)$. We can concatenate all the segments in $S'$ to obtain a new sequence $\pi'$. We then apply the shortest-path decoder on sequence $\pi$ to obtain a new solution $\mathrm{SP}(\pi')$. Since $S'$ corresponds to a path in the cost graph, whereas the solution $\mathrm{SP}(\pi')$ corresponds to a shortest path, the cost of $\mathrm{SP}(\pi')$ will never be higher than that of $\mathrm{SP}(\pi)$.

The second operator BIN-SHAKE preserves relative position of the bins and changes the relative position of some items inside each bin. Let $\mathrm{SP}(\pi)=(s_1,s_2,\ldots,s_k)$ be the solution produced by the SP decoder on sequence $\pi$. For each segment $s_i$, we randomly shuffle the elements in it to obtain a new segment $s_i'$. We then concatenate $s_1',s_2',\ldots,s_k'$ to obtain a new sequence $\pi'$. Let $\mathrm{SP}(\pi')$ be the solution found by the shortest-path decoder on sequence $\pi'$, following a similar argument as bin-shuffle, the cost of $\mathrm{SP}(\pi')$ will never be higher than that of $\mathrm{SP}(\pi)$.

The third operator SHUFFLE-SHAKE combines the previous two operators. It changes the relative position of some bins and the relative position of some items inside each bin. That is, given a sequence $\pi$ and its solution $\mathrm{SP}(\pi)$, the new sequence $\pi'$ is generated by randomly permuting the segments in $\pi$, and the items in each segment are randomly permuted. Once again, the solution produced by the SP decoder for this new sequence will be no worse than the original.

We use the example given by Table 1 to illustrate these three neighborhood search operators. The cost function is given in Fig. 1(b). Fig. 3(a) and (b) depicts the process of the bin-shuffle and bin-shake operators, respectively; the bins in the solutions for each sequence are demarcated by dotted lines.

Our iterative local search algorithm iteratively employs the SP decoder and neighborhood operators to explore multiple packing sequences (Algorithm 2).

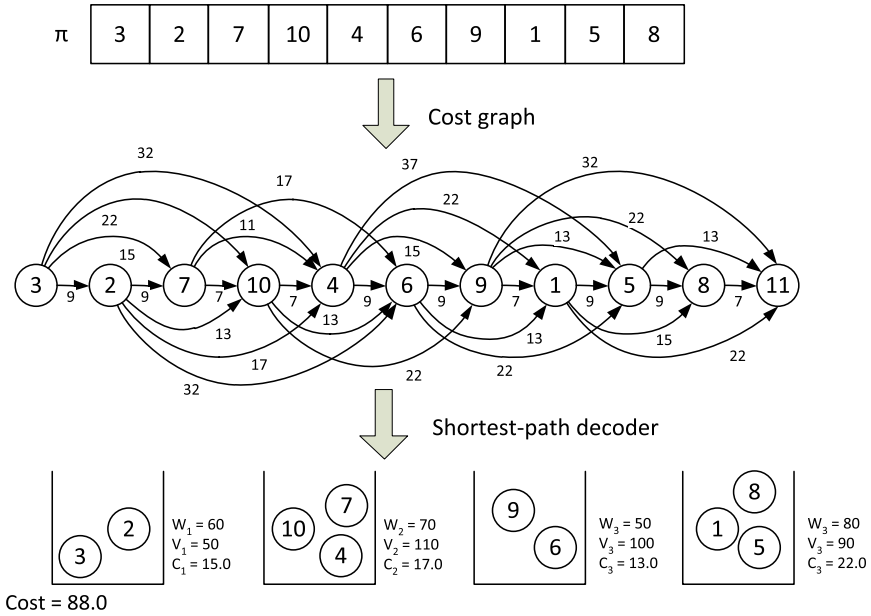


**Fig. 2.** Example of shortest-path decoder.

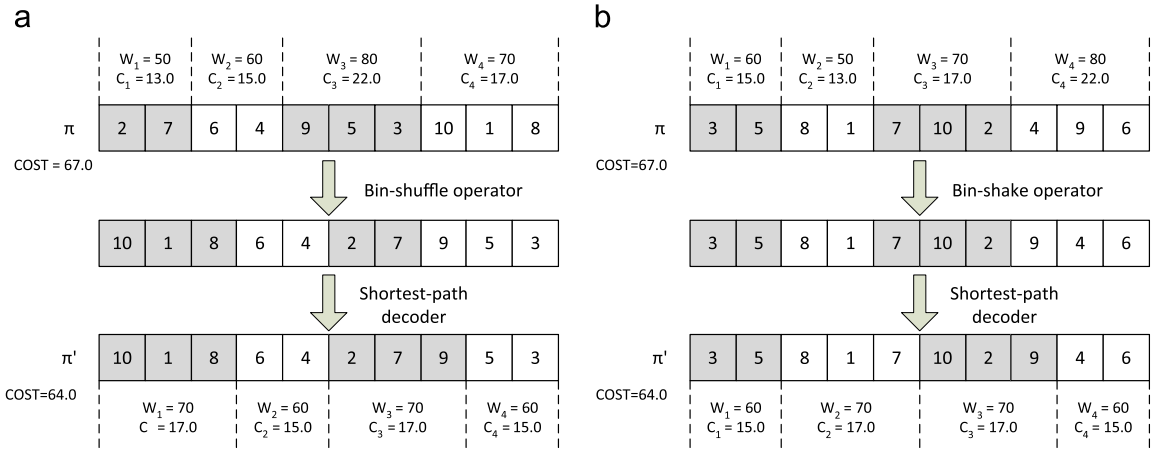


**Fig. 3.** (a) Bin-shuffle operator. (b) Bin-shake operator.

**Algorithm 2.** Iterative local search.

ILS (*iterCount*, *randSeed*)
// **Inputs**
// *iterCount*: number of iteration
// *randSeed*: seed for pseudo-random number generator
1 Initialize pseudo-random number generator with seed *randSeed*;
2 Generate a set of initial sequences $\Pi$ based on solutions produced by GWR;
3 **for** $i=1$ to *iterCount*
4 **for** each $\pi \in \Pi$
5 Apply the NeighborhoodSearch on $\pi$ to produce a new sequence $\pi'$;
6 **if** SP($\pi'$) costs less than SP($\pi$)
7 $\Pi = \Pi \cup \{\pi'\} \backslash \{\pi\}$;
8 **return** the best solution in $\Pi$;

The ILS begins by invoking the GWR procedure to obtain a set of initial solutions. For each initial solution, we obtain a sequence by concatenating the items in the bins together (the order of the items in each bin is decided arbitrarily).

Next, the ILS starts the iterative exploration in the solution space. For each sequence $\pi \in \Pi$, we employ the NeighborhoodSearch procedure on $\pi$ to produce a new sequence $\pi'$. The NeighborhoodSearch randomly selects a neighborhood operator from the bin-shuffle, bin-shake and shuffle-shake. The chosen operator produces a neighbor of $\pi$, which is denoted as $\pi'$. $\pi'$ is then decoded using the SP decoder. $\pi$ will be replaced by $\pi'$ in $\Pi$ if the corresponding solution of $S_B$ has a lower cost. This process is repeated *iterCount* times and finally the best solution in $\Pi$ is returned.

## 7. Computational experiments

Our algorithms are implemented as a sequential code in Java. The experimental results reported in this section are obtained on a Linux server equipped with an Intel Xeon E5430 CPU clocked at 2.66 GHz, with 8 GB RAM and running CentOS 5.4. The integer programming solver used is ILOG CPLEX 12.51 (64-bit edition). The test data and detailed results are available online at http://www.computational-logistics.org/orlib/2dvpp-plc.

### 7.1. Test instances

In order to evaluate our approach, we generated two categories of 2DVPP-PLC instances, **opt** and **rand**, according to the following schemes; the various parameters in the schemes were chosen after discussion with the manufacturer in question. For all instances, the cost function used is given by Fig. 1(b), and the capacity limit $V$ and weight limit $W$ of each bin are both set to 150.

The first category **opt** contains instances with known optimal solutions. Given the cost function $f(x)$ described in Expression (1), we can compute the average cost per unit weight $g(x)=f(x)/x$ (a plot of this function is given in Appendix C in the online supplements). The minimum of $g(x)$ is achieved when the bin weight is exactly 70 lb. Hence, if there exists a solution where the total weight of the items inside each bin is exactly 70 lb, then this packing plan must be optimal. Based on this observation, we generated the instances in this category as follows.

We take the number of bins $|B|$ from {25, 50, 100, 200}, resulting in four sets of instances. For each bin, we generate some items with total weight exactly equal to 70 and total volume less than 150 using the following process. The number of items to be constructed, denoted by *num*, is selected from the discrete uniform distribution $U_d[2,6]$. Next, we invoke the procedure Cut($num$, 70) (Algorithm 3) to generate the weight of each item; the procedure Cut($a$, $b$) produces an array with $a$ integer elements whose sum is $b$. To generate the volume of each item, we first select a value *total* from the discrete uniform distribution $U_d[70,150]$ and then invoke the procedure Cut($num$, $total$).

**Algorithm 3.** Generate positive integers with given sum.

Cut (*count*, *sum*)
1 $S$ = uniformly randomly generate *count* integers in range [1, *sum*];
2 $total$ = the sum of all integers in $S$;
3 Replace each $s \in S$ by $\lceil s\frac{sum}{total} \rceil$; // proportionally scale down
4 $excess = \sum_{s \in S} s - total$;
5 **while** $excess > 0$
6 randomly select an integer $s$ from $S$;
7 **if** $s > 1$, **then** reduce $s$ by 1 and $excess$ by 1;
8 **return** $S$;

If two randomly generated item types share the same weight and volume, we combine them into a single item type. For each instance set, we randomly generated 10 instances, for a total of 40

instances. All these instances have optimal solutions where the total weight of the items in each bin is 70. Each set is labeled **opt**$x$, where $x$ is the number of bins in the optimal solution.

The second category **rand** consists of four classes of instances where the total number of items $n \in \{128, 256, 512, 1024\}$. For each class, there are three sets of instances, for a total of 12 sets. In the three sets, there is an equal number of items of each type; the number of type $i$ items is $d_i \in \{1, 8, 32\}$. For all instances in this category, the weight and volume of each item are both selected from the discrete uniform distribution $U_d[1, 70]$. These instance sets are labeled **rand**$n-s$, where $n$ is the number of items and $s$ is the set index. In this category, there are 120 instances in total.

The 16 sets of instances are summarized in Appendix D in the online supplements.

### 7.2. Experiments with greedy-on-weight-range procedure

To solve an instance of the 2DVPP-PLC, our greedy-on-weight-range (GWR) procedure requires a decoder, a sorting rule and the number of artificial weight limits $L$. There are a total of 112 possible ways to configure GWR by picking a decoder from best-fit and first-fit, a sorting rule from WAVA, VAWA, WAVD, VAWD, WDVA, VDWA, WDVD and VDWD, and $L$ from {1, 2, 4, 8, 16, 32, 64}. For each configuration of GWR, we solve all 40 **opt** instances. For each instance, the percentage gap between the solution found by a configuration of GWR and the known optimal solution is computed. The average gap over all 40 **opt** instances is then taken as the performance measure for that configuration of GWR.

The results are summarized in Table 2, where each entry in the table is the performance of one configuration of GWR. The top half of the table corresponds to the best-fit decoder and the bottom half of the table corresponds to the first-fit decoder. GWR configurations with the same sorting rule are arranged in a column and configurations with the same $L$ are arranged in a row.

There are two interesting observations. Firstly, as expected, we can see that both sorting rules WDVD and VDWD outperform the other sorting rules regardless of the decoder used and values for $L$. The difference between WDVD and VDWD is mild. Hence we decide to use these two sorting rules to generate initial solutions. Secondly, the difference in the overall performance between first-fit and best-fit is very small for any given combination of sorting rule and $L$. Although the performance difference between best-fit and first-fit over the entire test set is almost zero, it does not mean the solutions produced for each instance are identical. Since both best-fit and first-fit are extremely fast, we decide to use both of them to create initial solutions.

From the design of GWR we know that increasing $L$ will never decrease solution quality. The computational results show that when $L$ is small, an increment in $L$ will lead to slightly improved solutions. However, when $L$ is large, further increment does not lead to any improvement in overall performance. When best-fit and first-fit are used with either sorting rule WDVD or VDWD, there is no observed improvement when $L > 4$. Hence, we decided to fix $L$ to 8.

### 7.3. Calibrating iterative local search

We have shown that both sorting rules WDVD and VDWD outperform other sorting rule while best-fit and first-fit produce different solutions with a similar quality. Therefore we generate four initial solutions using GWR to initialize $\Pi$ in our ILS algorithm (see Algorithm 2), one solution for each combination of decoder and sorting rule ($L$ is fixed to 8). We only need to decide a proper value for the parameter *iterCount* and a random seed *randSeed*. The random seed is used to initialize the pseudo-random number generator for generating random neighbors.

Instead of fixing *iterCount* to a constant for all test instances, we consider an adaptive strategy where *iterCount* scales linearly to the total number of items $n$ in an instance, that is, $iterCount = p \cdot n$. We conduct experiments on the **opt** instances to find the best value for $p$. For each test instance, we invoke ILS five times with *randSeed* taken from {3, 5, 7, 9, 11} and *iterCount* set to $256n$. For each invocation of ILS we record the best solution found so far at each iteration, and compute the percentage gap between the cost of the best solution and the known optimal solution. We use this percentage gap as the execution-level performance up to that iteration. For a specific iteration, we take the average of five executions for a test instance as instance-level performance up to that iteration. We move forward to compute the average of instance-level performance over a set of instances and call it set-level performance. In Fig. 4, we plot the set-level performance for test sets opt25, opt50, opt100, opt200 at specific iterations $pn$, $p \in \{0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, 128\}$. As we can see, the performance improves very quickly in the beginning and slows down as the iterations increases. After $8n$ iterations, the curve becomes pretty flat. We therefore decide to set $iterCount = 8n$ in all subsequent experiments.

Next, we investigate the fluctuation in performance due to randomness. For each instance, five solutions are obtained by ILS, one for each random seed. For each solution, the percentage gap to the known optimal solution is computed. The average gap over the five solutions is reported under the column *Avg. gap* (%) in Table 3. The variance is reported under the column *Variance*. The variance

**Table 2**
Average performance for GWR configurations on **opt** instances.

| Decoder | $L$ | WAVA (%) | VAWA (%) | WAVD (%) | VAWD (%) | WDVA (%) | VDWA (%) | WDVD (%) | VDWD (%) |
|---|---|---|---|---|---|---|---|---|---|
| BF | 1 | 4.94 | 2.35 | 5.18 | 2.61 | 1.78 | 1.66 | 1.00 | 0.98 |
| | 2 | 4.94 | 2.35 | 5.18 | 2.61 | 1.73 | 1.66 | 0.95 | 0.98 |
| | 4 | 4.94 | 2.35 | 5.18 | 2.61 | 1.68 | 1.66 | 0.94 | 0.98 |
| | 8 | 4.75 | 2.33 | 5.06 | 2.59 | 1.68 | 1.65 | 0.94 | 0.98 |
| | 16 | 4.53 | 2.33 | 4.83 | 2.59 | 1.68 | 1.65 | 0.94 | 0.98 |
| | 32 | 4.39 | 2.33 | 4.70 | 2.59 | 1.68 | 1.65 | 0.94 | 0.98 |
| | 64 | 4.39 | 2.33 | 4.70 | 2.59 | 1.68 | 1.65 | 0.94 | 0.98 |
| FF | 1 | 4.84 | 2.40 | 5.16 | 2.63 | 2.35 | 1.56 | 1.03 | 0.94 |
| | 2 | 4.84 | 2.39 | 5.16 | 2.57 | 2.32 | 1.56 | 0.98 | **0.93** |
| | 4 | 4.84 | 2.39 | 5.16 | 2.56 | 2.30 | 1.56 | 0.97 | **0.93** |
| | 8 | 4.66 | 2.37 | 5.04 | 2.56 | 2.30 | 1.54 | 0.97 | **0.93** |
| | 16 | 4.45 | 2.37 | 4.83 | 2.56 | 2.30 | 1.54 | 0.97 | **0.93** |
| | 32 | 4.32 | 2.37 | 4.70 | 2.56 | 2.30 | 1.54 | 0.97 | **0.93** |
| | 64 | 4.32 | 2.37 | 4.70 | 2.56 | 2.30 | 1.54 | 0.97 | **0.93** |

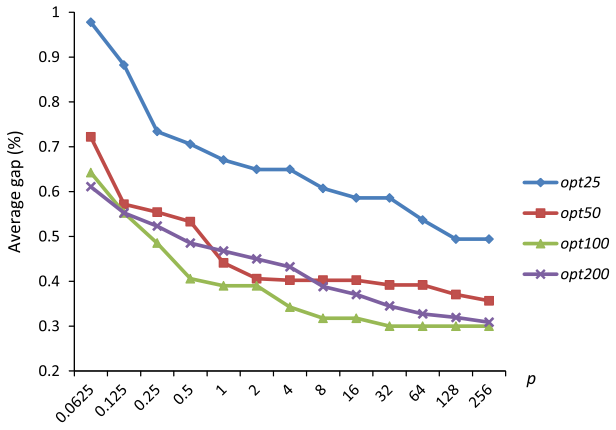


**Fig. 4.** Average performance of ILS over iterations.

**Table 3**
Effect of different random seeds in the ILS.

| Set | Instance | Avg. gap (%) | Variance | Set | Instance | Avg. gap (%) | Variance |
|---|---|---|---|---|---|---|---|
| opt25 | opt25-0 | 0.11 | 0.001 | opt100 | opt100-0 | 0.32 | 0.006 |
| | opt25-1 | 0.47 | 0.049 | | opt100-1 | 0.49 | 0.006 |
| | opt25-2 | 0.71 | 0.000 | | opt100-2 | 0.28 | 0.009 |
| | opt25-3 | 0.71 | 0.000 | | opt100-3 | 0.39 | 0.006 |
| | opt25-4 | 0.71 | 0.000 | | opt100-4 | 0.35 | 0.000 |
| | opt25-5 | 0.71 | 0.000 | | opt100-5 | 0.18 | 0.000 |
| | opt25-6 | 0.71 | 0.000 | | opt100-6 | 0.35 | 0.000 |
| | opt25-7 | 0.71 | 0.000 | | opt100-7 | 0.42 | 0.009 |
| | opt25-8 | 0.71 | 0.000 | | opt100-8 | 0.42 | 0.009 |
| | opt25-9 | 0.61 | 0.049 | | opt100-9 | 0.35 | 0.000 |
| opt50 | opt50-0 | 0.64 | 0.025 | opt200 | opt200-0 | 0.38 | 0.001 |
| | opt50-1 | 0.45 | 0.022 | | opt200-1 | 0.37 | 0.002 |
| | opt50-2 | 0.42 | 0.007 | | opt200-2 | 0.37 | 0.002 |
| | opt50-3 | 0.35 | 0.000 | | opt200-3 | 0.41 | 0.002 |
| | opt50-4 | 0.35 | 0.000 | | opt200-4 | 0.39 | 0.002 |
| | opt50-5 | 0.35 | 0.000 | | opt200-5 | 0.35 | 0.004 |
| | opt50-6 | 0.35 | 0.000 | | opt200-6 | 0.46 | 0.002 |
| | opt50-7 | 0.35 | 0.000 | | opt200-7 | 0.35 | 0.004 |
| | opt50-8 | 0.35 | 0.000 | | opt200-8 | 0.35 | 0.004 |
| | opt50-9 | 0.32 | 0.004 | | opt200-9 | 0.48 | 0.002 |

is very small compared to the average gap, which suggests that the fluctuation due to randomness is very small. Therefore we arbitrarily fix the random seed to 3 and only execute our ILS once for each instance when reporting our final results.

### 7.4. Experiments with the IP formulation

We develop two methods for solving the integer programming model (2)–(8). In the first method, M1, we solve the model directly using the commercial ILOG CPLEX solver version 12.51 with default setting. In the second method, M2, we make use of the heuristic callback feature of CPLEX solver. During the branch and bound tree search procedure, whenever a viable node is encountered, CPLEX will invoke a user supplied heuristic callback to produce a feasible solution. A node is viable if the LP relaxation of the node costs less than best known solution. In this case, it is possible that the heuristic callback may produce a feasible solution that is better than the best known solution and speedup the entire tree search. We adapt our ILS as follows and supply it as the heuristic callback. We fix $randseed = 3$ and the number of iterations $iterCount = 2n$ so that this version of ILS will produce a feasible solution quickly which is very important as it is invoked many times during the branch and bound tree search.

For bin packing instances with multiple identical items, it is well known that aggregating identical items results in equivalent but more efficient model. We replace constraints (3) and (7) (by 9) and (10), respectively:

$$\sum_{j \in B} \sum_{k \in K} x_{i,j}^k \geq d_i, \quad \forall i \in I \tag{9}$$

$$0 \leq x_{i,j}^k \leq d_i \text{ and integer}, \quad \forall i \in I,\ j \in B,\ k \in K \tag{10}$$

The aggregation is done for both methods. In our IP model, we need to know the number of bins in advance. For an **opt** instance, the number of bins is set to the number of bins in the known optimal solution. For a **rand** instance, the number of bins is set to the number of items.

We fix random seed used by CPLEX to 3, so that we can reproduce the results exactly should we repeat the experiments. In both methods, we set a time limit of 3600 s for each test instance. Since CPLEX checks time limit only after a tree node is solved, it is possible for a hard instance the root node itself takes hours to solve. In this case, we will stop after the first node is solved and report the total time used.

The comparison on the **opt** instances are summarized in Table 4. The column *#inst* gives the number of instances in an instance set. The column *#feasible* gives the number of instances that are successfully solved, and the column *#optimal* gives the number of instances with an optimal solution found and proven. The column *Avg. gap (%)* gives the percentage gap between the best solution found and the known optimal solution averaged over the solved instances in the test set. The column *Avg. time (seconds)* reports the average computational time over the instances that are successfully solved.

The results show that both M1 and M2 are capable to find optimal solutions for the majority of small instances in the test sets opt25 and opt50, where the known optimal solutions require only 25 and 50 bins, respectively. Strengthened by our ILS, M2 finds four more optimal solutions than M1 in the test set opt50. In addition, M2 takes much less computational time to prove the optimality for small instances. As the instance size grows larger, both methods fail to find any optimal solution within 1 h for opt100 and opt200 instances.

From Table 4, we can see that the solutions found by M2 on average have a much smaller gap to the optimal than M1. This is significant for practice use, as finding high-quality solutions is often more important that proving optimality.

The comparison on the **rand** instances are summarized in Table 5, with similar columns as Table 4.

For small instances from rand128-2, rand128-3, rand256-2 and rand256-3, M1 outperforms M2 in terms of average percentage gap. M2 cannot enumerate nodes as fast as M1 due to the additional time spend on heuristic callback. For instances with moderate and large size, the additional time spend on heuristic callback improves the best known solution and therefore enables

**Table 4**
Results by exact methods on **opt** instances (bold indicates better results).

| Method | Set | #inst | #feasible | #optimal | Avg. gap (%) | Avg. time (s) |
|---|---|---|---|---|---|---|
| M1 | opt25 | 10 | 10 | 10 | 0 | 180 |
| | opt50 | 10 | 10 | 2 | 0.22 | 3231 |
| | opt100 | 10 | 10 | 0 | 1.73 | 3600 |
| | opt200 | 10 | 10 | 0 | 3.92 | 3600 |
| M2 | opt25 | 10 | 10 | 10 | 0 | 89 |
| | opt50 | 10 | 10 | 6 | **0.14** | 2073 |
| | opt100 | 10 | 10 | 0 | **0.18** | 3600 |
| | opt200 | 10 | 10 | 0 | **0.14** | 4153 |

**Table 5**
Results by exact methods on **rand** instances (bold indicates better results).

| Set | # Inst | M1 | | | | M2 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | #feasible | #optimal | Avg. gap (%) | Avg. time (s) | #feasible | #optimal | Avg. gap (%) | Avg. time (s) |
| rand128-1 | 10 | 10 | 0 | 1.15 | 3600 | 10 | 0 | **0.66** | 3600 |
| rand128-2 | 10 | 10 | 0 | **0.82** | 3601 | 10 | 0 | 1.05 | 3413 |
| rand128-3 | 10 | 10 | 7 | **0.31** | 1083 | 10 | 8 | 0.39 | 749 |
| rand256-1 | 10 | 10 | 0 | 2.58 | 3600 | 10 | 0 | **0.57** | 3601 |
| rand256-2 | 10 | 10 | 0 | **1.02** | 3600 | 10 | 0 | 1.24 | 3600 |
| rand256-3 | 10 | 10 | 5 | **0.35** | 1924 | 10 | 4 | 0.64 | 2203 |
| rand512-1 | 10 | 10 | 0 | 7.58 | 3600 | 10 | 0 | **0.85** | 6453 |
| rand512-2 | 10 | 10 | 0 | 1.53 | 3600 | 10 | 0 | **1.13** | 3601 |
| rand512-3 | 10 | 10 | 0 | 0.97 | 3600 | 10 | 0 | **0.88** | 3600 |
| rand1024-1 | 10 | 2 | 0 | **24.80** | 3604 | 0 | 0 | N/A | > 3600 |
| rand1024-2 | 10 | 3 | 0 | 14.58 | 3600 | 10 | 0 | **1.10** | 3906 |
| rand1024-3 | 10 | 8 | 0 | 1.62 | 3600 | 10 | 0 | **0.99** | 3601 |

**Table 6**
Comparison on **opt** instances (bold indicates the best).

| Set | # Inst | GWR | | ILS | | M2 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | AG (%) | AT (s) | AG (%) | AT (s) | # Feasible | # Optimal | AG (%) | AT (s) |
| opt25 | 10 | 1.03 | 0.0 | 0.61 | 0.4 | 10 | 10 | **0.00** | 88.5 |
| opt50 | 10 | 0.80 | 0.0 | 0.39 | 2.9 | 10 | 6 | **0.14** | 2073.0 |
| opt100 | 10 | 0.71 | 0.0 | 0.37 | 22.2 | 10 | 0 | **0.18** | 3600.7 |
| opt200 | 10 | 0.66 | 0.1 | 0.40 | 240.0 | 10 | 0 | **0.14** | 4153.4 |
| Grand total | 40 | 0.80 | 0.0 | 0.42 | 25.8 | 40 | 16 | **0.11** | 2478.9 |

us to prune more branches in the branch and bound tree. M2 results in better overall performance than M1.

### 7.5. Overall comparisons

We proceed to present the overall results of our three different approaches: GWR, ILS and M2. In GWR, each instance is solved using the best four configurations, using best-fit and first-fit decoders with the sorting rules WDVD and VDWD, with $L=8$. For each instance we report the best solution and the total execution time of the four configurations. In ILS, each instance is solved once with random seed $randSeed=3$ and maximum iterations $iterCount=8n$, where $n$ is the number of items in an instance. In M2, each instance is solved once with random seed of CPLEX fixed to 3 and time limit set to 3600 s.

Table 6 reports the results for **opt** instances. The detailed results at instance level are included in Appendix E in the online supplements. The column *#inst* gives the number of instances in a test set. The columns under the heading *GWR*, *ILS* and *M2* report the performance of GWR, ILS and M2, respectively. The columns *AG* (%) report the percentage gap between the best solution found and the known optimal solution averaged over the solved instances in the test set. The columns *AT (s)* report the total CPU time in seconds averaged over the solved instances. For M2, the number of instances successfully solved is reported in the column *#feasible* and the number of instances with an optimal solution found and proven is given in the column *#optimal*. For GWR and ILS, all instances are successfully solved but none of optimal solutions are found.

As a simple heuristic, GWR is always extremely fast. It takes only a split of a second to solve each instance. As a result, GWR is a good choice when quick decisions are needed. For example, GWR could work as a subroutine in a larger decision support system where the bin packing problem is to be solved repeatedly. It can also be used in a nearly online scenario, that is, the input information is available only shortly before the decision to be made.

There is no surprise that ILS produces better solutions than GWR for all test sets, since ILS starts with the solutions found by GWR and iteratively improves them. The performance improvement by ILS ranges from 0.26% for opt200 instances to 0.42% for opt25 instances. Considering the fact that ILS only adds a few minutes of computational time and it never decreases the solution quality, it is still useful in many application scenarios.

The exact approach M2 achieves better solutions than GWR and ILS at the expense of computational time. Compared to ILS, the performance improvement by M2 is 0.31% in average. Instead of a single run of ILS, M2 invokes the ILS in the heuristic callback a number of times. From a different perspective, M2 can be viewed as a special multi-restart heuristic which frequently restarts the ILS procedure. Hence, there is also no surprise that M2 finds the best solution for all test sets.

Table 7 reports the results of GWR, ILS and M2 on the **rand** instances. The detailed results at instance level are included in Appendix E in the online supplements. The columns *Total cost* report the cost of the best solution summed over all instances in a test set. The columns *AT* (*s*) report the total CPU time in seconds averaged over all instances in a test set. The columns *AG* (%) report the percentage gap between the best solution found and the best lower bound achieved by M2 averaged over the solved instances in the test set. For instances in rand1024-1, which are failed to be solved by M2 within the given time, their lower bounds are obtained by computing their LP relaxations instead.

GWR can produce solutions in a split of a second for instances with up to 1024 items. ILS can slightly improve the solutions found by GWR for all test sets with a few minutes of additional CPU time per instance. Unlike **opt** instances, M2 does not outperform ILS for each **rand** set. Since M2 takes the number of items as an input to initialize the number of bins, this trivial bound on the number of bins brings in disadvantages that take away the efficiency. Still, M2 can obtain better results on the test sets rand128-1, rand128-2, rand128-3, rand256-2, rand256-3 and rand512-3, in which the number of item types is relatively small. For the other test sets, ILS is capable to achieve better solutions requiring much less computational time.

## 8. Conclusions and future work

In this paper, we introduced a new and practical two-dimensional vector packing problem with a piecewise linear cost function (2DVPP-PLC) based on the price structure from an actual courier company. It models the problem of shipping a number of

**Table 7**
Comparison on **rand** instances (bold indicates the best).

| Set | GWR | | | ILS | | | M2 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Total cost | AG (%) | AT (s) | Total cost | AG (%) | AT (s) | Total cost | AG (%) | AT (s) |
| rand128-1 | 11 094.2 | 0.79 | 0.00 | 11 081.6 | 0.68 | 0.46 | 11 080.1 | **0.66** | 3600.30 |
| rand128-2 | 10 651.9 | 1.45 | 0.00 | 10 620.7 | 1.19 | 0.46 | 10 607.5 | **1.05** | 3412.72 |
| rand128-3 | 11 734.8 | 0.88 | 0.02 | 11 713.2 | 0.59 | 0.52 | 11 700.6 | **0.39** | 748.77 |
| rand256-1 | 21 951.4 | 0.44 | 0.01 | 21 943.3 | **0.41** | 3.50 | 21 978.1 | 0.57 | 3600.56 |
| rand256-2 | 22 096.2 | 1.54 | 0.01 | 22 058.1 | 1.37 | 3.44 | 22 032.6 | **1.24** | 3600.22 |
| rand256-3 | 22 564.2 | 1.08 | 0.01 | 22 477.2 | 0.73 | 3.42 | 22 460.4 | **0.64** | 2202.99 |
| rand512-1 | 43 695.6 | 0.41 | 0.04 | 43 675.5 | **0.37** | 25.39 | 43 885.8 | 0.85 | 6452.64 |
| rand512-2 | 44 664.2 | 1.24 | 0.04 | 44 590.2 | **1.08** | 25.46 | 44 612.7 | 1.13 | 3600.70 |
| rand512-3 | 44 101.8 | 1.25 | 0.04 | 43 998.9 | 0.99 | 26.10 | 43 952.7 | **0.88** | 3600.23 |
| rand1024-1 | 87 394.4 | 0.25 | 0.15 | 87 386.6 | **0.24** | 282.33 | N/A | N/A | N/A |
| rand1024-2 | 88 088.6 | 0.96 | 0.15 | 87 959.3 | **0.81** | 282.38 | 88 205 | 1.10 | 3905.69 |
| rand1024-3 | 87 697.6 | 1.04 | 0.15 | 87 631.6 | **0.97** | 282.04 | 87 646.9 | 0.99 | 3600.67 |
| Grand total[a] | 408 340.5 | 1.01 | 0.04 | 407 749.6 | **0.83** | 59.38 | 408 162.4 | 0.86 | 3484.14 |
| Grand total[b] | 495 734.9 | 0.94 | 0.05 | 495 136.2 | **0.78** | 77.96 | N/A | N/A | N/A |

[a] Overall performance on the 11 instance sets, where rand1024-1 is excluded.
[b] Overall performance on all the 12 instance sets.

items in standard-sized cartons where the cost of a carton is a piecewise linear function of the weight of the items in the carton. We presented an integer programming formulation for this problem. Solving the model directly using commercial ILOG CPLEX is viable only for small instances. For large instances, we proposed a greedy-weight-range (GWR) procedure which addressed the shortcoming of the classical first-fit and best-fit heuristics. Computational results suggest that GWR can produce high-quality solutions in a split of a second for instances of practical size.

We adapted a shortest-path decoder to convert a sequence of items into a solution. Unlike the first-fit and best-fit decoder, the shortest-path decoder is complete for the 2DVPP-PLC, such that we can reduce the 2DVPP-PLC to the problem of searching a permutation of items. Such reduction greatly reduces the size of search space. We built an iterative local search (ILS) algorithm upon the shortest-path decoder to improve the solutions found by GWR. Computational results demonstrate improved solution quality for test instances. It is useful both in practice and as building block for designing exact algorithms to the 2DVPP-PLC.

A potential avenue of research is to design exact algorithms for the 2DVPP-PLC using more advanced techniques. For example, devising bounds on the number of bins required in optimal solutions and lower bounds for the 2DVPP-PLC. We have begun preliminary work on designing a branch-and-price algorithm for the 2DVPP-PLC.

## Acknowledgments

The authors thank an associate editor and two anonymous reviewers, whose comments greatly improved the quality of the paper.

## Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.omega.2014.07.004.

## References

[1] Anily S, Bramel J, Simchi-Levi D. Worst-case analysis of heuristics for the bin packing problem with general cost structures. Operations Research 1994;42:287–98.
[2] Arbib C, Marinelli F. On cutting stock with due dates. Omega 2014;46:11–20.
[3] Caprara A, Toth P. Lower bounds and algorithms for the 2-dimensional vector packing problem. Discrete Applied Mathematics 2001;111:231–62.
[4] Chan LMA, Muriel A, Shen Z-JM, Simchi-Levi D, Teo C-P. Effective zero-inventory-ordering policies for the single-warehouse multiretailer problem with piecewise linear cost structures. Management Science 2002;48:1446–60.
[5] Che CH, Huang W, Lim A, Zhu W. The multiple container loading cost minimization problem. European Journal of Operational Research 2011;214:501–11.
[6] Croxton KL, Gendron B, Magnanti TL. A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems. Management Science 2003;49:1268–73.
[7] Croxton KL, Gendron B, Magnanti TL. Variable disaggregation in network flow problems with piecewise linear costs. Operations Research 2007;55:146–57.
[8] Cui Y. Heuristic for the cutting and purchasing decisions of multiple metal coils. Omega 2014;46:117–25.
[9] Dyckhoff H. A typology of cutting and packing problems. European Journal of Operational Research 1990;44:145–59.
[10] Epstein L, Levin A. Bin packing with general cost structures. Mathematical Programming 2012;132:355–91.
[11] Eso M, Ghosh S, Kalagnanam J, Ladanyi L. Bid evaluation in procurement auctions with piecewise linear supply curves. Journal of Heuristics 2005;11:147–73.
[12] Fleszar K, Charalambous C. Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. European Journal of Operational Research 2011;210:176–84.
[13] Gendreau M, Laporte G, Semet F. Heuristics and lower bounds for the bin packing problem with conflicts. Computers & Operations Research 2004;31:347–58.
[14] Haouari M, Serairi M. Heuristics for the variable sized bin-packing problem. Computers & Operations Research 2009;36:2877–84.
[15] Hemmelmayr V, Schmid V, Blum C. Variable neighbourhood search for the variable sized bin packing problem. Computers & Operations Research 2012;39:1097–108.
[16] Kameshwaran S, Narahari Y. Nonconvex piecewise linear knapsack problems. European Journal of Operational Research 2009;192:56–68.
[17] Kang J, Park S. Algorithms for the variable sized bin packing problem. European Journal of Operational Research 2003;147:365–72.
[18] Kellerer H, Kotov V. An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing. Operations Research Letters 2003;31:35–41.
[19] Leung JY-T, Li C-L. An asymptotic approximation scheme for the concave cost bin packing problem. European Journal of Operational Research 2008;191:582–6.
[20] Liao C-S, Hsu C-H. New lower bounds for the three-dimensional orthogonal bin packing problem. European Journal of Operational Research 2013;225:244–52.
[21] Lim A, Rodrigues B, Wang Y. A multi-faced buildup algorithm for three-dimensional packing problems. Omega 2003;31:471–81.
[22] Lodi A, Martello S, Monaci M. Two-dimensional packing problems: a survey. European Journal of Operational Research 2002;141:241–52.
[23] Malaguti E, Durn RM, Toth P. Approaches to real world two-dimensional cutting problems. Omega 2014;47:99–115.
[24] Muritiba AEF, Iori M, Malaguti E, Toth P. Algorithms for the bin packing problem with conflicts. INFORMS Journal on Computing 2010;22:401–15.
[25] Smithin T, Harrison P. The third dimension of two-dimensional cutting. Omega 1982;10:81–7.
[26] Spieksma FC. A branch-and-bound algorithm for the two-dimensional vector packing problem. Computers & Operations Research 1994;21:19–25.

[27] Vielma JP, Ahmed S, Nemhauser G. Mixed-integer models for nonseparable piecewise-linear optimization: unifying framework and extensions. Operations Research 2010;58:303–15.
[28] Wei L, Oon W-C, Zhu W, Lim A. A goal-driven approach to the 2D bin packing and variable-sized bin packing problems. European Journal of Operational Research 2013;224:110–21.
[29] Woeginger GJ. There is no asymptotic ptas for two-dimensional vector packing. Information Processing Letters 1997;64:293–7.
[30] Wscher G, Hauner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research 2007;183:1109–30.
[31] Zhu W, Huang W, Lim A. A prototype column generation strategy for the multiple container loading problem. European Journal of Operational Research 2012;223:27–39.
[32] Zhu W, Zhang Z, Oon W-C, Lim A. Space defragmentation for packing problems. European Journal of Operational Research 2012;222:452–63.