European Journal of Operational Research 281 (2020) 25–35

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A branch-and-price algorithm for the two-dimensional vector packing problem

Lijun Wei[d], Minghui Lai[b], Andrew Lim[c], Qian Hu[a,*]

[a] School of Management and Engineering, Nanjing University, Nanjing 210093, China
[b] School of Economics and Management, Southeast University, Nanjing 211189, China
[c] Department of Industrial Systems Engineering and Management, National University of Singapore, Singapore
[d] Guangdong Provincial Key Laboratory of Computer Integrated Manufacturing System, State Key Laboratory of Precision Electronic Manufacturing Technology and Equipment, Guangdong University of Technology, Guangzhou 510006, China

ARTICLE INFO

Article history:
Received 2 August 2016
Accepted 12 August 2019
Available online 14 August 2019

Keywords:
Packing
Two-dimensional vector packing problem
Exact algorithm
Branch-and-price

ABSTRACT

The two-dimensional vector packing problem is a well-known generalization of the classical bin packing problem. It considers two attributes for each item and bin. Two capacity constraints must be satisfied in a feasible packing solution for each bin. The objective is to minimize the number of bins used. To compute optimal solutions for the problem, we propose a new branch-and-price algorithm. A goal cut that sets a lower bound to the objective is used. It is effective in speeding up column generation by reducing the number of iterations. To efficiently solve the pricing problem, we develop a branch-and-bound method with dynamic programming, which first eliminates conflicts between two items through branching, and then solves the two-constraint knapsack problem at leaf nodes through dynamic programming. Extensive computational experiments were conducted based on 400 test instances from existing literature. Our algorithm significantly outperformed the existing branch-and-price algorithms. Most of the test instances were solved within just a few seconds.

1. Introduction

In the two-dimensional vector packing problem (2DVPP), a set of n items and a set of identical bins are given. Let $I = \{1, 2, \ldots, n\}$ be the index set of the items. Each item $i \in I$ has two attributes: a weight w_i and a volume v_i. Each bin has a weight limit W and a volume capacity V. The 2DVPP is the problem of packing all the items into bins such that the number of used bins is minimized, satisfying the conditions that the total weight and the total volume of the items in each bin do not exceed W and V, respectively. Because of the two capacity constraints, the 2DVPP is also known as the two-constraint bin packing problem (Monaci & Toth, 2006). The 2DVPP generalizes the classical one-dimensional bin packing problem (BPP) by considering one additional attribute and the associated capacity constraints. Therefore, the 2DVPP, which is reduced to the BPP when $V = |I|$ and $v_i = 1$ for $i \in I$, is *NP*-hard. Without loss of generality, we assume that $0 < w_i \leq W$ and $0 < v_i \leq V$ for $i \in I$.

The 2DVPP has many applications in various areas such as packing and scheduling. In Chang, Hwang, and Park (2005), the problem of packing steel products, known as coils, into special bins known as cassettes, was investigated. The two attributes of each coil are the weight and size. In the case study of Billaut, Della Croce, and Grosso (2015), which considered a scheduling problem from the production of chemotherapy drugs for cancer treatment by intravenous injection, the 2DVPP is a special case of the scheduling problem when there are no due dates. Hu, Lim, and Zhu (2015) considered a shipping problem faced by a manufacturer that produces textile and clothes. The products are packed into identical cartons, and shipped to stores across the globe using a courier service. Because the textile products are flexible, only the weight and volume attributes are considered in the packing. The delivery cost is determined by the weight of a carton. Therefore, the shipping problem is a variant of the 2DVPP, and has a cost structure. Instead of minimizing the number of bins, the shipping problem minimizes the total delivery cost. The cost structure investigated in Hu et al. (2015) involves a piecewise linear cost function, and is extended to be general costs in Hu, Wei, and Lim (2018).

In the literature related to the 2DVPP, the existing work has mainly focused on lower bounds (Alves, de Carvalho, Clautiaux,

* Corresponding author.
E-mail addresses: villagerwei@gmail.com (L. Wei), laimh@seu.edu.cn (M. Lai), alim.china@gmail.com (A. Lim), huqian@nju.edu.cn (Q. Hu).

https://doi.org/10.1016/j.ejor.2019.08.024
0377-2217/

& Rietz, 2014; Caprara & Toth, 2001), approximation algorithms (Woeginger, 1997), and heuristics (Billaut et al., 2015; Masson et al., 2013; Monaci & Toth, 2006). Woeginger (1997) proved that there is no asymptotic polynomial time approximation scheme unless $P = NP$. In Kellerer and Kotov (2003), an approximation algorithm was proposed for the 2DVPP that has a worst case performance ratio of 2. The study in Monaci and Toth (2006) developed a column generation based heuristic to produce feasible solutions of good quality. In Masson et al. (2013), a multi-start iterated local search heuristic for bin packing problems was proposed and tested on the 2DVPP instances. In Billaut et al. (2015), two heuristics were proposed to solve a single machine scheduling problem with two-dimensional vector packing constraints. Recently, Buljubašić and Vasquez (2016) proposed a consistent neighborhood search heuristic for the BPP and the 2DVPP. Buljubašić and Vasquez (2016) efficiently solved 390 out of the 400 2DVPP instances.

Exact algorithms, especially branch-and-price algorithms, are popular methods for many combinatorial optimizations, including the BPP (Belov & Scheithauer, 2006; Brandão & Pedroso, 2016; de Carvalho, 1999; Delorme, Iori, & Martello, 2016) and many of its variants (Elhedhli, Li, Gzara, & Naoum-Sawaya, 2011; Pisinger & Sigurd, 2007; Sadykov & Vanderbeck, 2013). For the 2DVPP, Spieksma (1994) and Caprara and Toth (2001) studied branch-and-bound algorithms and a branch-and-price algorithm, respectively. A branch-and-price method for the 2DVPP was recently implemented in Alves et al. (2014). In Alves et al. (2014), a number of dual feasible functions to produce faster lower bounds were investigated. Brandão and Pedroso (2016) proposed an exact method for a set of bin packing problems, using a graph compression algorithm based on the arc-flow formulation, and solved 330 out of the 400 2DVPP instances to optimality. The study in Hu, Zhu, Qin, and Lim (2017) designed a branch-and-price algorithm that branches on a set of items for the 2DVPP with a piecewise linear cost function that minimizes the total cost. Recently, Heler, Gschwind, and Irnich (2018) proposed a stabilized branch-and-price algorithm using dual optimal inequalities for vector packing problems and solved 390 out of the 400 2DVPP instances to optimality.

In this work, for the 2DVPP, we propose a new branch-and-price algorithm that uses a goal cut to improve the lower bound and a branch-and-bound method with dynamic programming to generate columns. The 2DVPP is formulated using a set-partitioning formulation, which can be solved by a branch-and-price method (Section 2). A goal cut that sets a lower bound to the objective is proposed, to improve the lower bound at each node and speed up the column generation (Section 3). We investigate two pricing methods to generate columns with negative reduced cost (Section 4). The first pricing method is a branch-and-bound method that branches on a single item and obtains columns at leaf nodes. We propose an alternative branch-and-bound method, which only branches on two incompatible items. The problem at a leaf node is a two-constraint knapsack problem and solved by an accelerated dynamic programming method. An extensive computational analysis on 400 2DVPP test instances from the literature was conducted (Section 5). The results show that the goal cut is effective in speeding up the branch-and-price algorithm, and the branch-and-bound method with dynamic programming is more efficient in generating columns with negative reduced cost than the other pricing method. The proposed branch-and-price algorithm outperformed the existing branch-and-price methods for the 2DVPP. Most test instances were solved within a few seconds and five new best solutions were found.

2. Branch-and-price algorithm

We first derive a set-partitioning formulation (Section 2.1) for the 2DVPP. Based on the formulation, a branch-and-price method for the problem is introduced. In particular, we describe the branching scheme (Section 2.2), introduce notations (Section 2.3), formulate the master problem and the pricing problem in column generation (Section 2.4), and give a primal heuristic (Section 2.5). To improve the performance of the branch-and-price algorithm, we will propose a goal cut and design pricing methods in separate sections.

2.1. Set-partitioning formulation

A *packing pattern* describes a set of items that can be loaded into one bin without violating any capacity constraints. Let $\mathscr{P}$ be the index set of all the feasible packing patterns. A pattern $p \in \mathscr{P}$ is expressed by a vector $\mathbf{a}_p = (a_{1p}, a_{2p}, \ldots, a_{np})^T$, where $a_{ip} = 1$ if item i is in the packing pattern and $a_{ip} = 0$ otherwise. Let λ_p be a binary variable that equals to 1 if and only if pattern p is in the optimal solution of the 2DVPP and 0 otherwise. The set-partitioning formulation of the 2DVPP, denoted by F, is derived as follows:

$$(F) \quad z(F) = \min \quad \sum_{p \in \mathscr{P}} \lambda_p \tag{1}$$

$$\text{s.t.} \quad \sum_{p \in \mathscr{P}} a_{ip} \lambda_p = 1, \quad \forall i \in I \tag{2}$$

$$\lambda_p \in \{0, 1\}, \quad \forall p \in \mathscr{P}$$

The objective (1) is to minimize the number of bins. Constraints (2) ensure that each item is packed into exactly one bin. Let LF be the linear programming (LP) relaxation of formulation F and $z(LF)$ be the optimal value.

Formulation F is an integer programming (IP) problem, and the number of variables is exponential. Such a problem can be solved by a *branch-and-price* method, which enumerates subproblems through branch-and-bound to obtain the optimal integer feasible solution and solves the LP relaxation of the subproblem at each node by column generation.

2.2. Branching scheme (Ryan & Foster, 1981)

In the branch-and-price algorithm, we use the classical branching scheme proposed by Ryan and Foster (1981). The branching scheme has also been successfully used in Elhedhli et al. (2011) and Sadykov and Vanderbeck (2013) for the bin packing problem with conflicts.

For any two different items i and j, define $P(i, j)$ as a set including all the feasible patterns that contain both of the items, *i.e.*, $P(i, j) = \{p \in \mathscr{P} : a_{ip} = 1, a_{jp} = 1\}$. In the branch-and-price algorithm, if a node is not pruned and the optimal solution $\boldsymbol{\lambda}^*$ of the LP relaxation of its subproblem is fractional, the node is branched. Under the branching scheme, we first pick two different items i and j satisfying that the total value of patterns from $P(i, j)$ is also fractional, *i.e.*, $0 < \sum_{p \in P(i,j)} \lambda_p^* < 1$. Then, two new branches are created. On the first branch, the new subproblem enforces that the two items i and j must be loaded into the same bin. The branching constraint $\sum_{p \in P(i,j)} \lambda_p \geq 1$ is included. On the second branch, the new subproblem requires that the two items i and j must be loaded into different bins. The branching constraint $\sum_{p \in P(i,j)} \lambda_p = 0$ is included. In this case, items i and j are said to be *incompatible* or *in conflict*.

It is unnecessary to explicitly maintain the branching constraints in the subproblem at each node. The subproblem can be rendered by just removing inappropriate patterns and forbidding the generation of such inappropriate patterns. For the first subproblem with $\sum_{p \in P(i,j)} \lambda_p \geq 1$, the patterns that have either item i or item j are inappropriate; for the second subproblem with $\sum_{p \in P(i,j)} \lambda_p = 0$, the patterns that include both items i and j are inappropriate.

2.3. Notations

Formally, at a node, we denote by S the set of pairs of two different items which must be loaded into the same bin, and denote by T the set of pairs of two different items which must be loaded into different bins. The node is thus characterized by (S, T). In particular, the root node is denoted by $(\emptyset, \emptyset)$.

Since the branching constraints divide the patterns into different sets, we introduce notations to define some featured sets of patterns or items. The sets are helpful in the elaboration of our algorithms. Let $D \subseteq I$ be a set of items. The notations are summarized as follows:

- $P(i, j)$: the set of all the patterns that contain both items i and j, *i.e.*, $P(i, j) = \{p \in \mathscr{P} : a_{ip} = 1, a_{jp} = 1\}$;
- $Q(i, j)$: the set of all the patterns that include either item i or item j, *i.e.*, $Q(i, j) = \{p \in \mathscr{P} : a_{ip} + a_{jp} = 1\}$;
- $\mathscr{P}(S, T)$: the set of all the feasible patterns that satisfy the branching constraints at node (S, T), *i.e.*, $\mathscr{P}(S, T) = \mathscr{P} \setminus \big((\bigcup_{(i,j)\in T} P(i, j)) \cup (\bigcup_{(i,j)\in S} Q(i, j))\big)$.
- M_i: the set including item i and all the *companion items* that are forced to be loaded into the same bin with the item, *i.e.*, $M_i = \{j \in I : (i, j) \in S\} \cup \{i\}$;
- N_i: the set including all the *incompatible items* that are in conflict with any item from M_i, *i.e.*, $N_i = \{j \in I : (j, k) \in T,\ \forall k \in M_i\}$;
- $R(i, D)$: the set of items that are from $\{i+1, i+2, \ldots, n\}$ and compatible with all the items from D, *i.e.*, $R(i, D) = \{i+1, i+2, \ldots, n\} \setminus (\bigcup_{j\in D} N_j)$.

2.4. Column generation

For the subproblem at each node, its LP relaxation is solved by column generation. In the column generation, the *restricted master problem* (RMP) and the *pricing problem* are iteratively solved. We describe the mathematical formulations for the problems.

At node (S, T), the set of branching constraints specifies a set of inappropriate patterns which are eliminated from formulation F. All the feasible patterns are included into $\mathscr{P}(S, T)$. The *master problem*, which is the LP relaxation of the subproblem at the node, denoted by $SP(S, T)$, is given as follows:

$$(SP(S,T)) \quad \min \quad \sum_{p\in\mathscr{P}(S,T)} \lambda_p \tag{3}$$

$$\text{s.t.} \quad \sum_{p\in\mathscr{P}(S,T)} a_{ip}\lambda_p = 1, \quad \forall i \in I \tag{4}$$

$$0 \le \lambda_p \le 1, \quad \forall p \in \mathscr{P}(S, T) \tag{5}$$

Unlike the master problem, the RMP only includes a subset of columns (patterns). The pricing problem is solved to find columns with negative reduced cost. The columns are then added to the RMP to improve its solution. Let $\pi_i (\forall i \in I)$ be the dual variables associated with constraints (4). Let y_i be binary variables that equal to 1 if and only if item $i \in I$ is included into the generating pattern and 0 otherwise. The formulation of the pricing problem is given:

$$(2KPC(S,T)) \quad \min \quad 1 - \sum_{i\in I} \pi_i y_i \tag{6}$$

$$\text{s.t.} \quad \sum_{i\in I} w_i y_i \le W \tag{7}$$

$$\sum_{i\in I} v_i y_i \le V \tag{8}$$

$$y_i + y_j \le 1, \quad \forall (i, j) \in T \tag{9}$$

$$y_i = y_j, \quad \forall (i, j) \in S \tag{10}$$

$$y_i \in \{0, 1\}, \quad \forall i \in I \tag{11}$$

The objective (6) is to minimize the reduced cost. Constraints (7) and (8) ensure that the total weight and the total volume of the items in the generating column do not exceed the weight limit and volume capacity, respectively. Constraints (9) ensure that at most one item can be included for any two incompatible items. Constraints (10) ensure that the two given items must be loaded in the same bin according to the branching scheme. Constraints (10) can be removed if the two items are aggregated into a larger one. The formulation implies the two-constraint knapsack problem with conflicts (2KPC).

Initially, in the RMP, the set of columns includes all the elementary packing patterns that only load one item into a bin. With the elementary patterns and all the patterns inherited from parent node, the RMP at each node always has a basic feasible solution if it is feasible. In general, when infeasibility is observed, it is necessary to verify whether the new branching constraint leads the RMP to be infeasible or the current set of columns is insufficient to produce a basic feasible solution. In the latter case, one must introduce artificial variables and solve the auxiliary problem to find an initial basic feasible solution. With the elementary patterns, such checks are just saved.

2.5. Primal heuristic

Our branch-and-price algorithm enumerates nodes using depth-first search (DFS), which is helpful in improving the upper bound. After the LP relaxation of the subproblem at a node is solved by column generation, a heuristic is applied to obtain an integer feasible solution if the LP solution $\boldsymbol{\lambda}^*$ is not integral.

In the heuristic, a feasible solution is first constructed by including the patterns $p \in \mathscr{P}$ with $\lambda_p^* > 0.5$ and opening new bins for the remaining items. The items are packed into new bins using a first fit strategy, which picks an item of larger weight with a higher priority. The solution is improved by a local search procedure using two common operators: RELOCATE and SWAP. The RELOCATE operator moves an item from one bin to another bin; the SWAP operator exchanges two items from two different bins.

Let B_i be the set of items in bin i of the solution. Let $w(B_i)$ and $v(B_i)$ be the total weight and total volume of the items in B_i, respectively. The fitness of B_i is defined by the remaining capacity of the bin, *i.e.*, $f(B_i) = W - w(B_i) + V - v(B_i)$. The fitness is better is $f(B_i)$ is smaller. With fitness defined, the operators are improved to be more effective with similar implementations in Hu et al. (2018). The RELOCATE operator only moves an item from a bin wih worse fitness to another bin with better fitness. The EXCHANGE operator swaps two items only if the bin with better fitness can be further improved. In this way, the bins with worse fitness are expected to be eliminated so that the number of bins would be reduced.

3. Goal cut and early termination rules in column generation

To improve the performance of the branch-and-price algorithm, we propose a goal cut (Section 3.1) and early termination rules (Section 3.2) to accelerate the column generation.

3.1. Goal cut

The 2DVPP aims to minimize the number of bins in the objective and the optimal value is always an integer. At any node in the branch-and-bound tree, if the optimal value of the master problem is fractional, then the smallest integer that is greater than the value is a valid lower bound. By enforcing the objective to be no less than the lower bound, we obtain a cardinality inequality, as shown in Lemma 1. We call it a *goal cut*, as it constrains the objective.

Lemma 1. *If lb is a lower bound of the subproblem at node (S, T), then the following inequality is valid:*

$$\sum_{p\in\mathscr{P}(S,T)} \lambda_p \geq \lceil lb \rceil \tag{12}$$

The goal cut is general, and can be extended to many problems as long as their objectives have an integer value. For some other problems where a minimum increment exists in the objective, the goal cut can also be adapted. Though the idea is simple, the goal cut indeed has many desirable features.

First, the goal cut is simple but effective. It improves the lower bound at each node, and the improved lower bound is helpful in improving the performance of the branch-and-price algorithm.

Second, absence of separation and ease of maintenance are additional advantages of the goal cut. Since no further information except *lb* is required, no separation is needed for the goal cut. The goal cut can be included into the master problem before the root node is solved. When the algorithm moves on to solve a different node, the goal cut is conveniently modified by updating the right hand side (RHS) accordingly. For instance, at the root node, lb is initialized by the value of $\max\{\frac{1}{W}\sum_{i\in I} w_i, \frac{1}{V}\sum_{i\in I} v_i\}$. At any other node, lb is set to the lower bound of its parent node.

Third, the structure of the pricing problem is preserved. Let γ be the dual value of constraint (12). The pricing problem 2*KPC* is updated by replacing the objective (6) with (13). Unlike many other cuts and valid inequalities, the goal cut does not introduce any indicator constraints or additional variables into the pricing problem. The pricing methods remain applicable and effective without redesign.

$$\min \quad 1-\gamma-\sum_{i\in I}\pi_i y_i \tag{13}$$

Finally, and most importantly, the goal cut is effective in reducing the number of iterations in the column generation. At a child node, the RHS of the goal cut is set to $\lceil lb \rceil$, where lb is the lower bound of the parent node. Unlike many other valid inequalities that are unchangeable once added, the goal cut is flexible and dynamically improved. In the column generation, lb can be improved after a certain number of iterations, as shown in Section 3.2. The strengthened goal cut helps terminate column generation earlier.

3.2. Early termination rules in column generation

In the column generation, the RMP and the pricing problem are iteratively solved until no columns with negative reduced cost are found. After the column generation, the optimal value of the RMP is also the optimal value of the master problem. Let z^* be the optimal value of the master problem and $\bar{z}$ be the optimal value of the current RMP at an iteration in the column generation. Because more columns with negative reduced cost would exist and more iterations would be executed, the inequality $z^* \leq \bar{z}$ holds. Let $\bar{c}^*$ be the least reduced cost when the pricing problem is optimally solved at the iteration. For problems that optimize the cardinality in the objective, like $\sum_{p\in\mathscr{P}} \lambda_p$ in the 2DVPP for instance, Lübbecke and Desrosiers (2005) pointed out that the following inequality always holds:

$$\bar{z} + \bar{c}^* z^* \leq z^* \leq \bar{z}$$

That is, $\bar{z} + \bar{c}^* z^*$ is a valid lower bound for the master problem. Further, this implies a new lower bound:

$$\frac{\bar{z}}{1-\bar{c}^*} \leq z^* \tag{14}$$

The new lower bound is useful for early termination in column generation. Furthermore, we propose three rules (Lemma 2) to improve the lower bound of the master problem and decide early termination in the column generation. In this way, the column generation and hence the branch-and-price algorithm are sped up.

Lemma 2. *The following three rules are effective in improving the column generation at each node:*

(1) if $\lceil \frac{\bar{z}}{1-\bar{c}^*} \rceil > \lceil lb \rceil$, *then the goal cut is updated with* $lb = \lceil \frac{\bar{z}}{1-\bar{c}^*} \rceil$ *and the lower bound of the master problem is improved;*
(2) if $\lceil \frac{\bar{z}}{1-\bar{c}^*} \rceil$ *is greater than or equal to an upper bound of the 2DVPP, then the column generation is terminated and the node is pruned;*
(3) if $\lceil \frac{\bar{z}}{1-\bar{c}^*} \rceil = \lceil \bar{z} \rceil$, *then the goal cut is updated with* $lb = \lceil \bar{z} \rceil$. *The RMP is then solved again to obtain an optimal solution and the column generation is terminated.*

4. Pricing methods

The goal cut is helpful in reducing the number of iterations in the column generation. To reduce the computational time of each iteration and therefore improve the efficiency of the column generation, a fast pricing procedure plays an important role.

We introduce two branch-and-bound algorithms to solve the 2KPC. The first algorithm (Section 4.1) is adapted from the branch-and-bound method for the knapsack problem with conflicts (Sadykov & Vanderbeck, 2013). It branches on a single item so that feasible patterns are obtained at leaf nodes. Since it enumerate nodes with depth first search (DFS), we call it branch-and-bound with DFS (BnB-DFS). The second algorithm (Section 4.2) branches on two items that are in conflict. At a leaf node, no two incompatible items exist. The subproblem becomes the two-constraint knapsack problem and is solved by a dynamic programming method. We call the algorithm a branch-and-bound algorithm with dynamic programming (BnB-DP). Computational results (Section 5) demonstrate that BnB-DP is more efficient than BnB-DFS.

For the ease of describing the pricing methods, we here give some necessary notations. Let $\bar{c}$ be the upper bound of the pricing problem, *i.e.*, the reduced cost of a feasible column. Initially, $\bar{c}$ is set to 0 because only columns with negative reduced cost should be considered. Let D be the set of items that are chosen in a generating pattern and $\mathscr{C}$ be a set including the columns with negative reduced cost. Define $w(D) = \sum_{i\in D} w_i$, $v(D) = \sum_{i\in D} v_i$, and $c(D) = 1-\gamma-\sum_{i\in D}\pi_i$ as the total weight, the total volume, and the reduced cost of the set of items D, respectively. Furthermore, the items are sorted when the pricing procedure starts. For instance, they can be sorted in ascending order of τ_i, where τ_i is the average price per unit weight over all the companion items of item i, *i.e.*,

$$\tau_i = -\frac{\sum_{j\in M_i}\pi_j}{\sum_{j\in M_i} w_j}.$$

4.1. Branch-and-bound with DFS

The first branch-and-bound algorithm for the pricing problem is adapted from Sadykov and Vanderbeck (2013). It branches on a single item and explore nodes with depth first search (DFS), as shown in Algorithm 1 . We denote the algorithm by BnB-DFS. It is started by invoking BnB-DFS(0, ∅, ∅) to find columns with negative reduced cost.

In the algorithm, a feasible solution of the 2KPC is generated by iteratively including an item into the knapsack without violating the two capacity constraints. Due to the conflicts among the items, once an item is loaded, all its companion items are also loaded and all its incompatible items are removed from consideration. If D is the current pattern and item i is the last item included, the resulting problem, denoted as 2*KPC*(i, D), is to to fill the knapsack with the remaining compatible items from $R(i, D)$.

Algorithm 1: BnB − DFS($i, D, \mathscr{C}$)

Data: the upper bound $\bar{c}$, the set of chosen items D;
Result: return the set of columns $\mathscr{C}$;
if $c(D) < \bar{c}$ **then**
  $\bar{c} = c(D)$;
  $\mathscr{C} = \mathscr{C} \cup \{D\}$;
end
Compute a lower bound lb of $2KPC(i, D)$;
if $lb \geq \bar{c}$ **then**
  return $\mathscr{C}$;
end
$j \leftarrow i + 1$;
while $c(D) - (W - w)\tau_j < \bar{c}$ **and** $j \leq n$ **do**
  if $j \notin D$ **and** *item j is compatible with all items in D* **then**
    $D' = D \cup M_j$;
    if $w(D') \leq W$ **and** $v(D') \leq V$ **then**
      $\mathscr{C} \leftarrow$ BnB-DFS($j, D', \mathscr{C}$);
    end
  end
  $j \leftarrow i + 1$;
end
return $\mathscr{C}$.

The bounding procedure can reduce the number of nodes to be examined. The lower bound of $2KPC(i, D)$ is computed and used to prune nodes by comparing it to the current upper bound of the 2KPC. If the conflicts among the items are relaxed, the relaxation problem is the two-constraint knapsack problem (2KP). The 2KP can be solved exactly by dynamic programming to obtain a tight lower bound of $2KPC(i, D)$. However, the number runs of solving the 2KP would be extremely large. To obtain lower bounds of $2KPC(i, D)$ using dynamic programming would be inefficient. Instead, we use fast alternatives to compute lower bounds by solving the one-dimensional continuous knapsack problem.

Let λ and μ be two positive weights that are used to aggregate the two attributes of items and the knapsack. Particularly, each item $j \in R(i, D)$ has an aggregated attribute of $\lambda w_j + \mu v_j$ and the knapsack has a capacity of $\lambda(W - w(D)) + \mu(V - v(D))$. We denote the continuous knapsack problem by $KP(i, D, \lambda, \mu)$ and give its formulation in the following:

$$
\begin{aligned}
&(KP(i, D, \lambda, \mu)) \\
&\min \quad c(D) - \sum_{j \in R(i,D)} \pi_j y_j \\
&\text{s.t.} \quad \sum_{j \in R(i,D)} (\lambda w_j + \mu v_j) y_j \leq \lambda(W - w(D)) + \mu(V - v(D)) \\
&\qquad 0 \leq y_j \leq 1, \quad \forall j \in R(i, D)
\end{aligned}
$$

The continuous knapsack problem is solved by a greedy algorithm. Since the items are always sorted when the pricing procedure begins, the greedy algorithm runs just in $O(m)$, where $m = |R(i, D)|$. Though the lower bound is slightly worse than that obtained from the 2KP, this fast bounding procedure performs better in terms of efficiency. Moreover, we consider $(\lambda, \mu) \in \{(0, 1), (1, 0), (1, 1)\}$ to obtain three lower bounds and pick the best among them.

4.2. Branch-and-bound with dynamic programming

BnB-DFS decides an item through branching and obtains a feasible pattern at a leaf node. The performance of BnB-DFS depends on the number of nodes enumerated. We propose a new branch-and-bound method. It branches on two incompatible items each time so that there is no conflict among the items at a leaf node. The problem at a leaf node becomes the 2KP. Dynamic programming is applied to the 2KP to obtain feasible columns with negative reduced cost. The branch-and-bound algorithm with dynamic programming (BnB-DP) performs more efficiently than BnB-DFS for the 2DVPP.

BnB-DP mainly consist of two parts: a branch-and-bound framework for the 2KPC (Section 4.2.1) and a dynamic programming method for the 2KP (Section 4.2.2).

4.2.1. Branch-and-bound for the 2KPC

The framework of the BnB-DP is described in Algorithm 2. In the algorithm, D is a set including the chosen items in the generating pattern and E is a set including the forbidden items that are eliminated from consideration. If an item is incompatible with any item from D or forbidden during branching, it is included into E. Initially, the upper bound $\bar{c}$ is set to 0. BnB-DP is started by invoking BnB-DFS($\emptyset, \emptyset, \emptyset$).

Algorithm 2: BnB-DP($D, E, \mathscr{C}$)

Data: the upper bound $\bar{c}$, the set of chosen items D, the set of forbidden items E;
Result: return the set of columns $\mathscr{C}$;
Compute a lower bound lb of $2KPC(D, E)$;
if $lb \geq \bar{c}$ **then**
  return $\mathscr{C}$;
end
if *no conflict exists in* $I \setminus (D \cup E)$ **then**
  $\mathscr{C}' \leftarrow 2\text{KP} - \text{DP}(\bar{c}, D, E)$;
  Update the upper bound $\bar{c}$;
  return $\mathscr{C} \cup \mathscr{C}'$;
end
Pick two incompatible items i_1 and i_2 from $I \setminus (D \cup E)$;
$\mathscr{C}_1 \leftarrow \mathscr{C}_2 = \mathscr{C}_3 = \emptyset$;
$D' = D \cup M_{i_1}$; // case 1: $y_{i_1} = 1, y_{i_2} = 0$;
if $w(D') \leq W$ **and** $v(D') \leq V$ **then**
  $\mathscr{C}_1 \leftarrow$ BnB-DP($D', E \cup N_{i_1}, \mathscr{C}$);
end
$D' = D \cup M_{i_2}$; // case 2: $y_{i_1} = 0, y_{i_2} = 1$;
if $w(D') \leq W$ **and** $v(D') \leq V$ **then**
  $\mathscr{C}_2 \leftarrow$ BnB-DP($D', E \cup N_{i_2}, \mathscr{C}$);
end
$D' = D$; //case 3: $y_{i_1} = 0, y_{i_2} = 0$;
$\mathscr{C}_3 \leftarrow$ BnB-DP($D', E \cup M_{i_1} \cup M_{i_2}, \mathscr{C}$);
$\mathscr{C} = \mathscr{C} \cup \mathscr{C}_1 \cup \mathscr{C}_2 \cup \mathscr{C}_3$;
return $\mathscr{C}$;

Given the sets D and E, the problem at a node of BnB-DP, denoted by $2KPC(D, E)$, is to fill the knapsack with the remaining items from $I \setminus (D \cup E)$. In the bounding procedure, the lower bound of $2KPC(D, E)$ is computed by also solving a continuous knapsack problem using the greedy algorithm. If the lower bound is not less than $\bar{c}$, the current node is pruned. Otherwise, the node is branched if any two incompatible items exist; if no conflict exists, the subproblem of the node is hence the 2KP and solved by a dynamic programming procedure, denoted as $2KP - DP(\bar{c}, D, E)$.

In the branching, a pair of two incompatible items (i_1, i_2) is chosen. Three new subproblems are obtained by setting: (1) $y_{i_1} = 1,\ y_{i_2} = 0$; (2) $y_{i_1} = 0,\ y_{i_2} = 1$; or (3) $y_{i_1} = y_{i_2} = 0$. The chosen item and all its companion items are included into D; meanwhile, all its incompatible items are forbidden and added into E. For instance, in case (1), item i_1 is chosen and item i_2 is forbidden. The items from M_{i_1} including item i_1 and all its companion items are added into D.

The items from N_{i_1} which are incompatible with item i_1 are added into E. Note that item i_2 and all its companion items are already in N_{i_1}, *i.e.*, $M_{i_2} \subseteq N_{i_1}$.

4.2.2. Dynamic programming for the 2KP

To solve the 2KP at a leaf node, we propose an efficient dynamic programming method. Suppose the set of chosen items D and the set of forbidden items E are given. We denote by $2KP(i, w, v)$ the problem of filling the knapsack with the first i items from $I\backslash(D\cup E)$ so that exactly w units of weight and v units of volume are used in the knapsack, where $w \in \{0, 1, \ldots, W\}$ and $v \in \{0, 1, \ldots, V\}$. Let $J(i, w, v)$ be the solution value of $2KP(i, w, v)$ and $Y(i, w, v)$ be a binary variable that equals to 1 if item i is chosen in the solution and 0 otherwise. The dynamic programming recursions are given as follows:

$$J(i,w,v) = \min_{Y(i,w,v)\in\{0,1\}} \Big\{ J\Big(i-1,\ w - w(M_i)\ Y(i,w,v),\ v - v(M_i)\ Y(i,w,v)\Big) - Y(i,w,v) \sum_{j\in M_i} \pi_j \Big\}$$

$$J(0,w,v) = \infty$$

$$J(0,0,0) = c(D)$$

In the implementation, $J(i, w, v)$ is shortened to $J(w, v)$ to achieve space efficiency. In each iteration, the first item from $I\backslash(D\cup E)$ is picked. The item and its companion items from M_i are considered to update the states (w, v). When the iteration is done, the items from M_i are included into the forbidden set E to shorten the list of the remaining items. The time complexity of the dynamic programming algorithm is $O(mWV)$ with $m = |I \setminus (D \cup E)|$.

States (w, v) are said to be *active* if $J(w, v) \neq \infty$ and *inactive* if $J(w, v) = \infty$. In the state propagation, only active states are extended to the other states by adding items. Suppose the active state (w, v) is extended to state (w', v') by adding items from M_i, where $w' = w + w(M_i)$ and $v' = v + v(M_i)$. If $J(w', v')$ is reduced, state (w', v') is set to active and updated accordingly. Moreover, each active state implies a feasible solution. If the solution value is less than the upper bound $\bar{c}$, $\bar{c}$ is updated.

The lower bound of state (w', v') is obtained by first solving a continuous knapsack problem using the greedy method. In the problem, items from $R(i, D)\backslash(E\cup M_i)$ are available and the knapsack has a weight limit of $W - w(D) - w'$ and a volume capacity of $V - v(D) - v'$. For item $j \in R(i, D)\backslash(E\cup M_i)$, the cost is $-\pi_j$. The objective is to minimize the total cost. We denote by $lb_r(i, w', v')$ the optimal value of the continuous knapsack problem. The inequality $lb_r(i, w', v') \leq lb_r(i+1, w', v')$ holds because the items are already sorted. Suppose item i is now considered in the state propagation. The lower bounds of states (w', v') and (w, v) are updated with the following equations:

$$lb(w', v') = J(w'v') + lb_r(i, w', v') \tag{15}$$

$$lb(w, v) = J(w, v) - \sum_{j\in M_i} \pi_i + lb_r(i, w', v') \tag{16}$$

If the lower bound of a state is larger than $\bar{c}$, the state is set to inactive. By reducing the number of active states, the dynamic programming method is thus accelerated.

5. Computational experiments

The proposed algorithm was tested on the 2DVPP test instances from the literature. The algorithms were implemented using JAVA programming language. Computational results were obtained on a PC equipped with 8 gigabytes RAM and an Intel(R) Core(TM) i7-6700 CPU clocked at 3.40 gigahertz. The restricted master problems were sovled using the solver ILOG CPLEX 12.63 and only a single thread was set. The results can be found online via http://sites.google.com/site/orlib222/2dvpp/.

5.1. Test instances

In the literature, there are 400 2DVPP test instances that were proposed by Caprara and Toth (2001). The test instances are classified into 10 groups. In each group, the 40 test instances are divided into 4 sets according to the number of items. For the first 9 groups, the number of items is from {25,50,100,200}. For the last group, the number of items is from {24, 51, 99, 201}. For each set, there are 10 test instances that have the same number of items.

In Caprara and Toth (2001), two branch-and-bound algorithms and a branch-and-price algorithm were tested on the 2DVPP test instances. Results showed that their branch-and-price algorithm performed the best. Recently, Alves et al. (2014) investigated a number of dual feasible functions and lower bounds for the multidimensional vector packing problem. They pointed out that the lower bound computed by solving the LP relaxation of the set-partitioning formulation was the best among all the lower bounds. They also implemented a branch-and-price algorithm to compute the lower bounds and optimal solutions for the 2DVPP instances. To evaluate the performance of the exact algorithms for the 2DVPP, our branch-and-price algorithm was compared to the two branch-and-price algorithms on the 400 instances.

5.2. Analyses on the goal cut and pricing methods

We first conduct computational experiments to examine and analyze the goal cut and the two pricing methods. The four different implementations of the branch-and-price algorithm in the following were tested:

- *BP(BnB-DFS)*, the branch-and-price algorithm that invokes BnB-DFS to solve the pricing problem and uses no cut;
- *BP(BnB-DFS+GC)*, the branch-and-price algorithm that invokes BnB-DFS to solve the pricing problem and uses the goal cut;
- *BP(BnB-DP)*, the branch-and-price algorithm that invokes BnB-DP to solve the pricing problem and uses no cut;
- *BP(BnB-DP+GC)*, the branch-and-price algorithm that invokes BnB-DP to solve the pricing problem and uses the goal cut;

The four branch-and-price implementations were tested on the 100 instances that have at least 200 items. Two runs were executed for each implementation and each instance. In the first run, only the root node was solved by column generation with a time limit of 600 seconds. In the second run, the branch-and-bound tree was explored by branch-and-price with a time limit of 600 seconds. Table 1 and 2 give the comparison results of the four implementations.

In Table 1, the columns *class* and *n* show the group id and the number of items for the 10 instances of a set, respectively. For each implementation, the columns *#iter*, *t (s)*, t_m *(s)*, and t_p *(s)* give the number of iterations in column generation, the computational time of solving the root node, the computational time of solving the master problem, and the computational time of solving the pricing problem averaged over the 10 instances of a set, respectively. The columns *#sol* report the number of instances that were successfully solved in a set. For BP(BnB-DP) and BP(BnB-DP+GC), all the instances were successfully solved.

In Table 2, the columns *UB*, *LB*, and *#node* give the total of upper bounds, the total of lower bounds, and the total of explored nodes for the solved instances in a set, respectively. The columns *#opt* report the number of instances that were optimally solved and the columns *t (s)* report the computational time averaged over

Table 1
Comparison results of the four implementations at root node.

class	n	BP (BnB-DFS)					BP (BnB-DFS+GC)					BP (BnB-DP)				BP (BnB-DP+GC)			
		#sol	#iter	t (s)	t_m (s)	t_p (s)	#sol	#iter	t (s)	t_m (s)	t_p (s)	#iter	t (s)	t_m (s)	t_p (s)	#iter	t (s)	t_m (s)	t_p (s)
01	200	10	119.4	72.8	0.8	71.8	10	72.7	5.8	0.4	5.2	134	10.4	0.9	9.4	62.1	2.7	0.4	2.1
02	200	10	22.9	0.1	0.01	0.01	10	25.4	0.1	0.02	0.01	12.9	0.2	0.01	0.05	13.7	0.2	0.01	0.1
03	200	10	24.6	0.1	0.01	0.01	10	28.7	0.1	0.01	0.01	16.3	0.2	0.01	0.1	16.1	0.2	0.01	0.1
04	200	2	98	506.0	1.3	1198.8	8	53.4	128.4	0.6	224.9	106.4	21.2	1.5	19.6	37.4	4.4	0.5	3.8
05	200	3	167.1	473.8	6.9	872.8	10	1	0.1	0.01	0.01	138.3	23.1	3.3	19.6	1	0.01	0.002	0.001
06	200	10	82.5	0.7	0.1	0.4	10	76.3	0.4	0.1	0.2	99.5	0.5	0.1	0.2	90.6	0.5	0.1	0.2
07	200	10	171.5	1.1	0.3	0.6	10	96.2	0.4	0.1	0.1	218.8	0.8	0.3	0.3	91.3	0.4	0.1	0.1
08	200	10	23.7	0.2	0.02	0.1	10	23.7	0.1	0.01	0.1	15.4	0.1	0.003	0.04	15.6	0.1	0.01	0.03
09	200	10	147.9	200.8	1.1	199.5	10	120.4	64.9	0.9	63.8	225.9	20.2	1.3	18.7	128.8	9.9	1.0	8.7
10	201	10	106.7	1.5	0.2	1.1	10	106.7	1.5	0.2	1.1	164.9	0.8	0.3	0.3	165.3	0.9	0.4	0.4
All		85	96.43	125.7	1.1	234.5	98	60.45	20.2	0.2	29.5	113.24	7.7	0.8	6.8	62.19	1.9	0.3	1.5

Table 2
Comparison results of different branch-and-price implementations.

class	n	BP (BnB-DFS)					BP (BnB-DFS+GC)					BP (BnB-DP)					BP (BnB-DP+GC)				
		UB	LB	#opt	#node	t (s)	UB	LB	#opt	#node	t (s)	UB	LB	#opt	#node	t (s)	UB	LB	#opt	#node	t (s)
01	200	504	503	9	668	163.9	503	503	10	966	12.9	503	503	10	1114	36.4	503	503	10	1126	7.6
02	200	1135	1135	10	10	0.2	1135	1135	10	10	0.2	1135	1135	10	10	0.2	1135	1135	10	10	0.2
03	200	1135	1135	10	10	0.1	1135	1135	10	10	0.2	1135	1135	10	10	0.2	1135	1135	10	10	0.2
04	200	256	51	2	84	1210.4	253	201	8	84	144.8	253	253	10	84	22.5	253	253	10	84	5.4
05	200	130	39	3	10	812.7	130	130	10	10	0.02	130	130	10	10	22.4	130	130	10	10	0.03
06	200	811	811	10	652	2.6	811	811	10	430	1.2	811	811	10	526	1.9	811	811	10	534	1.4
07	200	801	801	10	152	1.4	801	801	10	166	0.7	801	801	10	116	1.0	801	801	10	242	0.8
08	200	1000	1000	10	10	0.2	1000	1000	10	10	0.2	1000	1000	10	10	0.1	1000	1000	10	10	0.1
09	200	513	503	0	748	602.5	513	503	0	4044	601.6	509	503	4	7910	418.3	513	503	0	14768	601.0
10	201	670	670	10	4252	27.9	670	670	10	5192	36.1	670	670	10	2732	8.7	670	670	10	5684	17.7
All		6955	6648	74	6596	282.2	6951	6889	88	10922	79.8	6947	6941	94	12522	51.2	6951	6941	90	22478	63.4

Table 3
Comparison results on the 2DVPP test instances (I).

class	n	BP(Caprara 2001)			BP(Alves et al 2014)			BP(BnB-DP+GC)				
		#node	#opt	t (s)	LB	#opt	t (s)	LB	UB	#opt	#node	t (s)
01	25	1	10	7.7	69	10	33.9	69	69	10	1	0.03
01	50	2	10	129.5	135	8	343	135	135	10	3.2	0.1
01	100	7	10	1579.8	255	0	600	255	255	10	29	1.3
01	200	1	9	19481.8	503	0	600	503	503	10	112.6	7.6
02	25	1	10	0.4	142	10	0.4	142	142	10	1	0.02
02	50	5	10	2.5	315	10	2.5	315	315	10	1	0.0
02	100	8	10	21.4	574	10	12.3	574	574	10	1	0.1
02	200	1	10	146.0	1135	10	422.2	1135	1135	10	1	0.2
03	25	1	10	0.4	142	10	0.5	142	142	10	1	0.02
03	50	4	10	2.6	315	10	1.8	315	315	10	1	0.04
03	100	7	10	10.0	569	10	3.7	569	569	10	1	0.1
03	200	12	10	64.4	1135	10	70.1	1135	1135	10	1	0.2
04	25	1	10	12.2	33	7	75.3	33	33	10	1	0.04
04	50	1	10	114.1	70	6	371.5	70	70	10	1	0.2
04	100	1	10	3197.1	130	0	600	130	130	10	1	0.02
04	200	1	3	7033.5	253	0	600	253	253	10	8.4	5.4
05	25	1	10	7.1	20	0	600	20	20	10	1	0.01
05	50	1	10	65.0	40	0	600	40	40	10	1	0.3
05	100	1	10	1344.0	70	0	600	70	70	10	1	0.01
05	200	1	8	40212.1	130	0	600	130	130	10	1	0.03

the 10 test instances in a set. If an instance was not successfully solved at the root node within the given time limit, its lower bound was set to zero. The computational time was recorded until the running iteration of the column generation was finished if the time limit was exceeded. Hence, some computational times could be larger than the given time limit.

It is shown in Table 1 that the goal cut is effective in reducing the number of iterations in the column generation. On average, the number of iterations is reduced by about 50% by the goal cut. It owes to the better lower bound and the early termination rules enabled by the goal cut. As the number of iterations is reduced in the column generation, the computational times on solving the master problem and the pricing problem are also reduced. Both BP(BnB-DFS) and BP(BnB-DP) are improved after adding the goal cut. BP(BnB-DFS+GC) solves 13 more instances at the root node within the time limit and the average computational time is significantly reduced. BP(BnB-DP+GC) runs much faster than the other three. Its average computational time is less than 2 seconds.

By comparing the pricing methods, Table 1 shows that BnB-DP is more efficient than BnB-DFS. Without the goal cut, BP(BnB-DP) runs much faster than BP(BnB-DFS) by about 15 times. With the goal cut, BP(BnB-DP+GC) is about 10 times faster than BP(BnB-DFS+GC) in terms of the average computational time. Moreover, the average computational time on solving the pricing problem by BnB-DFS is fluctuated significantly over different instance sets. Meanwhile, the average pricing time of BnB-DP is just no more than 20 seconds over the 100 instances with 200 or 201 items. As no conflicts exists in the pricing problem at the root node, the time complexity of BnB-DP depends on the polynomial-time dynamic programming method for the 2KP. The computational time is thus predictable. As to BnB-DFS, it searches a branch-and-bound tree with possibly a large depth. To implicitly compute the the large number of nodes in the branch-and-bound tree could be slow. BnB-DP is more stable and faster, and can be used in branch-and-price for various test instances.

As shown in Table 2, the implementations using BnB-DP outperformed those using BnB-DFS in terms of the number of optimal solutions and the average computational time. For BP(BnB-DFS), more optimal solutions are verified and the performance is significantly improved if the goal cut is used. The goal cut is also useful for BP(BNB-DP). The performance on many test sets is improved and the computational times are reduced.

5.3. Comparison results

Our branch-and-price algorithm BP(BnB-DP+GC) was compared to the two branch-and-price algorithms in the literature on all the instances. The time limit of each run was 600 CPU seconds. The branch-and-price algorithm proposed by Caprara and Toth (2001), denoted as BP(Caprara & Toth (2001)), was tested on a Digital DEC-Station 5000/240 CPU that has approximately the same speed as a PC 486/100 with a time limit of 10,000 CPU seconds for each run. The branch-and-price algorithm of Alves et al. (2014), denoted as BP(Alves et al. (2014)), was tested on a PC with an Intel Core i3 CPU with 2.27 gigahertz and 4 gigabytes of RAM; each run was set with a time limit of 600 CPU seconds.

The comparison results of the three algorithms on the 400 instances are summarized in Tables 3 and 4. The results are grouped into three blocks; each stands for an algorithm. The columns *class* and n give the group id and the number of items in each instance, respectively. In Caprara and Toth (2001), the average number of nodes, the number of test instances that were optimally solved, and the average computational time in CPU seconds were reported. They are shown in columns *#node*, *#opt*, and *t (s)*, respectively. In addition to columns *#opt* and *t (s)*, the total of the lower bounds (column *LB*) for the 10 instances of a set is reported in Alves et al. (2014). As to our algorithm, the additional column *UB* reports the sum of the upper bounds for the 10 instances of a set.

Our algorithm outperformed the two branch-and-price algorithms in terms of both the number of optimal solutions and the average computational time. 390 out of the 400 instances were optimally solved by our algorithm BP(BnB-DP+CC) and the average computational time was only a few seconds.

We also compared BP(BnB-DP+CC) to two state-of-the-art heuristics, which are the iterated local search heuristic (MS-ILS) proposed by Masson et al. (2013) and the consistent neighborhood search heuristic (CNS) developed by Buljubašić and Vasquez (2016). The MS-ILS was run with a time limit of 300 seconds on an Opteron 2.4 gigahertz with 4 gigabytes of RAM memory running Linux OpenSuse 11.1. The CNS was run with a time limit of 10 seconds on a computer with an Intel Core i7-3770 CPU 3.40 gigahertz processor, which is comparable to our machine. As the machine used by the MS-ILS is about five times slower (Buljubašić & Vasquez, 2016), the running times are divided by five. For each instance, the CNS was run 50 times with difference seeds. Table 5 re-

Table 4
Comparison results on the 2DVPP test instances (II).

class	n	BP(Caprara 2001)			BP(Alves et al 2014)			BP(BnB-DP+GC)				
		#node	#opt	t (s)	LB	#opt	t (s)	LB	UB	#opt	#node	t (s)
06	25	1	10	1.7	101	9	6.1	101	101	10	1.6	0.02
06	50	2	10	14.5	215	9	68.1	215	215	10	3.4	0.05
06	100	13	10	308.1	410	8	552.2	410	410	10	10	0.2
06	200	1	8	6565.5	811	0	600	811	811	10	53.4	1.4
07	25	1	10	2.3	96	10	8.1	96	96	10	1.8	0.01
07	50	7	10	16.7	197	10	87.2	197	197	10	2.8	0.03
07	100	22	10	320.2	402	6	607.9	402	402	10	1.8	0.1
07	200	31	10	4136.1	801	0	600	801	801	10	24.2	0.8
08	25	2	10	0.7	130	10	3.2	130	130	10	1	0.02
08	50	8	10	2.7	250	10	5.7	250	250	10	1	0.03
08	100	16	10	18.6	500	10	21.8	500	500	10	1	0.1
08	200	30	10	206.3	1000	10	140	1000	1000	10	1	0.1
09	25	1	10	7.5	73	8	24.5	73	73	10	1	0.1
09	50	1	10	244.2	145	9	416.6	145	145	10	1	0.1
09	100	1	10	5567.2	267	0	600	267	267	10	1	1.3
09	200	–	0	–	503	0	600	503	513	0	1476.8	601.0
10	24	1	10	2.0	80	10	1.5	80	80	10	1	0.02
10	51	1	10	16.0	170	10	5.5	170	170	10	1	0.1
10	99	1	10	290.4	330	9	30.1	330	330	10	1	0.2
10	201	–	0	–	670	0	600	670	670	10	568.4	17.7
All (1-10)		–	368	2398.7	13186	249	277.9	13186	13196	390	58.1	16.0

Table 5
Comparison results with heuristics.

n	Class	LB	MS-ILS		CNS			BP(BnB-DP+GC)	
			Best	Time (s)	Avg	Best	Time (s)	Best	Time (s)
25	01	69	69	2.54	69.0	69	0.00	69	0.03
25	06	101	101	4.26	101.0	101	2.00	101	0.02
25	07	96	96	3.72	96.0	96	1.00	96	0.01
25	09	73	73	4.06	73.0	73	10.00	73	0.05
24	10	80	80	2.22	80.0	80	0.00	80	0.02
50	01	135	135	14.50	135.0	135	0.00	135	0.08
50	06	215	215	13.70	215.0	215	2.00	215	0.05
50	07	197	197	17.60	197.0	197	1.00	197	0.03
50	09	145	145	39.80	145.0	145	10.00	145	0.13
51	10	170	170	13.80	170.0	170	0.00	170	0.06
100	01	255	257	58.90	255.0	255	0.03	255	1.26
100	06	410	410	60.00	410.0	410	5.00	410	0.20
100	07	402	402	57.80	402.0	402	4.00	402	0.12
100	09	267	267	60.00	267.0	267	10.00	267	1.28
99	10	330	330	46.40	330.0	330	0.01	330	0.22
200	01	503	503	60.00	503.0	503	0.01	503	7.63
200	06	811	811	60.00	811.0	811	8.00	811	1.41
200	07	801	802	60.00	801.0	801	2.00	801	0.84
200	09	503	513	60.00	513.0	513	10.01	513	600.96
201	10	670	678	60.00	670.1	670	0.91	670	17.66

ports the comparison results of the MS-ILS, the CNS, and BP(BnB-DP+CC) on the test instances from groups 01, 06, 07, 09, and 10. BP(BnB-DP+CC) was run only once for each instance with a time limit of 600 seconds. In the table, the columns *Best* and *Time (s)* report the total of the best upper bounds and the average computational time over the 10 instances of a set. For the CNS, the column *Avg* gives the average of the total upper bounds over the 50 runs.

Both the CNS and BP(BnB-DP+CC) solved 190 out of the 200 instances. None of the three methods computed an optimal solution for any instance with 200 items from group 09. Considering the efficiency in terms of the average computational time, BP(BnB-DP+CC) used significantly less CPU seconds on 11 out of the 20 sets as shown in Table 5. The CNS performed better on the instances from the groups 01 and 10.

To the best of our knowledge, the 10 instances with 200 items from group 09 have not been optimally solved. By extending the time limit to 3600 seconds, both BP(BnB-DP+GC) and BP(BnB-DP) were run on these instances for finding new best solutions. The results are summarized in Table 6. The column *Instance* shows the name of an instance, and the columns *LB*, *UB*, *#opt*, and *t*(*s*) report the lower bound, the upper bound, the number of verified optimal solutions, the computational time in CPU seconds for each instance, respectively.

As shown in Table 6, five new best solutions were computed by BP(BnB-DP). Though improved using the goal cut for efficiency, BP(BnB-DP+GC) only verified one optimal solution. Indeed, the goal cut is helpful in improving the lower bound and reduce the number of iterations of column generation at each node, but not directly related to upper bounding. We infer that the set of columns in the optimal solution of the master problem after column generation would matter. With the goal cut, the column generation terminates early and thus the solution of the master problem would be quite different from that of BP(BnB-DP). In the branching,

Table 6
Results on the hardest 2DVPP test instances and new best solutions.

Instance	LB	BP (BnB-DP)			BP (BnB-DP+GC)		
		UB	#opt	t (s)	UB	#opt	t (s)
CL_09_200_01	50	51	0	3601.1	51	0	3605.7
CL_09_200_02	50	51	0	3600.1	51	0	3600.3
CL_09_200_03	50	50	1	172.7	51	0	3600.3
CL_09_200_04	50	50	1	262.1	51	0	3600.6
CL_09_200_05	50	51	0	3601.0	51	0	3602.6
CL_09_200_06	50	51	0	3601.2	51	0	3600.4
CL_09_200_07	50	51	0	3600.8	51	0	3600.2
CL_09_200_08	51	51	1	70.6	52	0	3601.0
CL_09_200_09	51	51	1	77.2	51	1	3015.3
CL_09_200_10	51	51	1	800.2	52	0	3600.6

Table 7
Results on "hard28" BPP test instances.

Instance	n	LB	BP (BnB-DP + GC)			
			UB	#opt	#node	Time (s)
BPP119	200	77	77	1	3	1.0
BPP13	180	67	67	1	61	2.4
BPP14	160	62	62	1	2713	62.7
BPP144	200	73	73	1	1461	47.5
BPP175	200	84	84	1	59	1.9
BPP178	200	80	80	1	17	1.0
BPP181	180	72	72	1	1915	48.4
BPP195	180	64	64	1	819	22.6
BPP359	180	76	76	1	29,443	695.5
BPP360	160	62	62	1	37	1.0
BPP40	160	59	59	1	153	3.9
BPP419	200	80	80	1	239	7.2
BPP47	180	71	71	1	37	1.1
BPP485	180	71	71	1	7	0.7
BPP531	200	83	83	1	13	0.7
BPP561	200	72	72	1	1107	32.2
BPP60	160	63	63	1	3623	66.1
BPP640	180	74	74	1	55	1.3
BPP645	160	58	58	1	135	3.3
BPP709	180	67	67	1	61	2.3
BPP716	180	76	76	1	70,497	1754.3
BPP742	160	64	64	1	11	0.6
BPP766	160	62	62	1	21	0.9
BPP781	200	71	71	1	77	3.4
BPP785	180	68	68	1	113	3.2
BPP814	200	81	81	1	41	1.1
BPP832	160	60	60	1	63	1.6
BPP900	200	75	75	1	879	25.7
All		1972	1972	28	4059.3	99.8

different pairs of items would be selected by BP(BnB-DP+GC) and BP(BnB-DP). As a result, their branch-and-bound tree would be totally different. Moreover, our primal heuristic is to search a feasible solution based on the solution of the master problem. If any column in the solution is never used in an optimal 2DVPP solution, it would be hard for the heuristic to improve the upper bound.

As a whole, the goal cut is helpful in improving efficiency for branch-and-price. To futher improve the algorithm, dedicated branching rules and an independent heuristic that does not rely on the solution of the master problem could be possible. To optimally solve the remaining five instances remains a challenge. We tried to extend the time limit to 3 hours, but no more new best solutions were found.

Moreover, we tested BP(BnB-DP+GC) on 28 difficult BPP instances in the set "hard28". For each BPP instance, a 2DVPP instance was constructed by assigning $v_i = 1$ for $i \in I$ and $V = |I|$. The "hard28" instances were then solved by applying BP(BnB-DP+GC) to the new 2DVPP instances. For each run, BP(BnB-DP+GC) was set with a time limit of 3600 seconds. Table 7 show the results on the "hard28" BPP instances. All the instances were optimally solved and 26 out of the 28 test instances were solved in less than 70 seconds. As pointed out by Delorme et al. (2016), the best exact algorithm for the BPP is the branch-and-cut-and-price algorithm by Belov and Scheithauer (2006). According to the computational results in Delorme et al. (2016), the branch-and-cut-and-price algorithm is the only algorithm among the tested ones that can optimally solve all the "hard28" instances in less than one minute on a machine with an Intel Xeon 3.10 gigahertz processor and 8 gigabyte RAM. It should be noted that the performance of BP(BnB-DP+GC) can be further improved for the BPP by reducing dimensions, optimizing the codes, and introducing strong structural cuts like Belov and Scheithauer (2006).

6. Conclusions

In this paper, we proposed a branch-and-price algorithm BP(BnP-DP+GC) for the 2DVPP, which uses a branch-and-bound algorithm with dynamic programming to solve the pricing problem and a goal cut. The goal cut is effective in improving the lower bound at each node and reducing the number of iterations in the column generation. As the column generation is faster, the branch-and-price algorithm becomes more efficient. Two branch-and-bound algorithms for the pricing problem were investigated. The first one is BnB-DFS, which is adapted based on the branch-and-bound algorithm given in the literature for the knapsack problem with conflicts. BnB-DFS branches on any item, and returns a column with negative reduced cost at a leaf node. Instead, BnB-DP branches on two incompatible items and solves the problem at each leaf node using an efficient dynamic programming method. As shown by our computational results, BnB-DP is more efficient than BnB-DFS. Furthermore, our exact algorithm outperformed the existing branch-and-price algorithms for the 2DVPP. In addition, five new best solutions were reported.

Still, there remain five 2DVPP instances that were not optimally solved. It remains a challenge to solve these in future work. The branch-and-price method using a goal cut can be applied to many other combinatorial optimization problems that have a set-partitioning or set-covering formulation. In future work, the algorithm could be improved by incorporating problem-based valid inequalities and the synergies among the goal cut, the branching scheme and the primal heuristic can be studied.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (Grants 71571094, 71871070, 71501039, 71732003, 71531010), and Science and Technology Planning Project of Guangdong Province of China (Grant 2019A050503010), and partially supported by NRF Singapore (Grant NRF-RSS2016-004), MOE-AcRF-Tier 2 (Grant MOE2017-T2-2-153).

References

Alves, C., de Carvalho, J. V., Clautiaux, F., & Rietz, J. (2014). Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem. *European Journal of Operational Research, 233*(1), 43–63.

Belov, G., & Scheithauer, G. (2006). A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research, 171*(1), 85–106.

Billaut, J.-C., Della Croce, F., & Grosso, A. (2015). A single machine scheduling problem with two-dimensional vector packing constraints. *European Journal of Operational Research, 243*(1), 75–81.

Brandão, F., & Pedroso, J. P. (2016). Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research, 69*, 56–67.

Buljubašić, M., & Vasquez, M. (2016). Consistent neighborhood search for one-dimensional bin packing and two-dimensional vector packing. *Computers & Operations Research, 76*, 12–21.

Caprara, A., & Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics, 111*(3), 231–262.

de Carvalho, J. V. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research, 86*, 629–659.

Chang, S. Y., Hwang, H.-C., & Park, S. (2005). A two-dimensional vector packing model for the efficient use of coil cassettes. *Computers & operations research, 32*(8), 2051–2058.

Delorme, M., Iori, M., & Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research, 255*(1), 1–20.

Elhedhli, S., Li, L., Gzara, M., & Naoum-Sawaya, J. (2011). A branch-and-price algorithm for the bin packing problem with conflicts. *INFORMS Journal on Computing, 23*(3), 404–415.

Heler, K., Gschwind, T., & Irnich, S. (2018). Stabilized branch-and-price algorithms for vector packing problems. *European Journal of Operational Research, 271*(2), 401–419. doi:10.1016/j.ejor.2018.04.047.

Hu, Q., Lim, A., & Zhu, W. (2015). The two-dimensional vector packing problem with piecewise linear cost function. *Omega, 50*, 43–53.

Hu, Q., Wei, L., & Lim, A. (2018). The two-dimensional vector packing problem with general costs. *Omega, 74*, 59–69.

Hu, Q., Zhu, W., Qin, H., & Lim, A. (2017). A branch-and-price algorithm for the two-dimensional vector packing problem with piecewise linear cost function. *European Journal of Operational Research, 260*(1), 70–80.

Kellerer, H., & Kotov, V. (2003). An approximation algorithm with absolute worst–case performance ratio 2 for two-dimensional vector packing. *Operations Research Letters, 31*(1), 35–41.

Lübbecke, M. E., & Desrosiers, J. (2005). Selected topics in column generation. *Operations Research, 53*(6), 1007–1023.

Masson, R., Vidal, T., Michallet, J., Penna, P. H. V., Petrucci, V., Subramanian, A., & Dubedout, H. (2013). An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. *Expert Systems with Applications, 40*(13), 5266–5275.

Monaci, M., & Toth, P. (2006). A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing, 18*(1), 71–85.

Pisinger, D., & Sigurd, M. (2007). Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing, 19*(1), 36–51.

Ryan, D. M., & Foster, B. A. (1981). An integer programming approach to scheduling. In A. Wren (Ed.), *Computer scheduling of public transport urban passenger vehicle and crew scheduling* (pp. 269–280). Amsterdam: North-Holland Publishing Company.

Sadykov, R., & Vanderbeck, F. (2013). Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing, 25*(2), 244–255.

Spieksma, F. C. (1994). A branch-and-bound algorithm for the two-dimensional vector packing problem. *Computers & operations research, 21*(1), 19–25.

Woeginger, G. J. (1997). There is no asymptotic ptas for two-dimensional vector packing. *Information Processing Letters, 64*(6), 293–297.