VPSolver 3: Multiple-choice Vector Packing Solver

Filipe Brandão

INESC TEC and Faculdade de Ciências, Universidade do Porto, Portugal

`fdabrandao@dcc.fc.up.pt`

February 17, 2016

Abstract

VPSolver is a vector packing solver based on an arc-flow formulation with graph compression. In this paper, we present the algorithm introduced in VPSolver 3.0.0 for building compressed arc-flow models for the multiple-choice vector packing problem.

Keywords: Multiple-choice Vector Bin Packing, Arc-flow Formulation, Integer Programming.

1 Introduction

The vector bin packing problem (VBP), also called general assignment problem by some authors, is a generalization of bin packing with multiple constraints. In this problem, we are required to pack n items of m different types, represented by p-dimensional vectors, into as few bins as possible. The multiple-choice vector bin packing problem (MVP) is a variant of VBP in which bins have several types (i.e., sizes and costs) and items have several incarnations (i.e., will take one of several possible sizes).

Brandão and Pedroso (2016) present a general arc-flow formulation with graph compression for vector packing. This formulation is equivalent to the model of Gilmore and Gomory (1961), thus providing a very strong linear relaxation; the largest absolute gap found in all the instances solved in Brandão and Pedroso (2016) was 1.0027. Given a directed acyclic multi-graph containing every valid packing pattern represented as a path from a source node to a target node, the general arc-flow formulation is equivalent to the Gilmore-Gomory's model with the same set of patterns as those represented as paths in the graph.

In Brandão and Pedroso (2016), a large variety of applications through reductions to vector packing is presented. Brandão and Pedroso (2013) extends this method to multiple-choice vector bin packing problems by building a compressed arc-flow graph for each bin type. In this method, a super source node is connected to the source node of each arc-flow graph and each target node is connected to a super target. Since every feasible packing pattern using any of the available bin types is represented as a path from the super source to the super target, the general arc-flow formulation can be applied.

In this paper, we present an alternative algorithm for building compressed arc-flow graphs for multiple-choice vector packing problems. This algorithm is a generalization of the direct Step-3 graph construction algorithm proposed in Brandão and Pedroso (2016). If there is only one bin type and each item has a single incarnation, the new algorithm produces exactly the same graph as the one produced by the original algorithm. When multiple bin types exist, the algorithm produces on a single run a graph containing all the valid packing patterns for all bin types. This new algorithm is usually much faster than the method proposed in Brandão and Pedroso (2013), and it usually produces smaller graphs.

The remainder of this paper is organized as follows. Section 2 presents the arc-flow formulation for MVP. In Section 3, we show how to derive arc-flow models from dynamic programming recursions, and how to obtain smaller models using graph compression. Finally, Section 4 introduces the new graph construction and compression algorithm, and Section 5 presents some conclusions.

2 General arc-flow formulation for MVP

For a given i, let $\mathbf{J}_i$ be the set of incarnations of item i, and let $\mathbf{I} = \{(i,j) : i = 1, \ldots, m, \; j \in \mathbf{J}_i\}$ be the set of items. Let $(i,j) \in \mathbf{I}$ be the incarnation j of item i, and w_{ij} its weight vector. For the sake of simplicity, we define $(0,0)$ as an item with weight zero in every dimension; this artificial item is used to label loss arcs. Let b_i be the demand of items of type i, for $i = 1, \ldots, m$. Let b_0 be the total number of items (i.e., $b_0 = \sum_{i=1}^m b_i$). Let q be the number of bin types, and let W_t and C_t be the capacity vector and the cost of bins of type t, respectively.

Given a directed acyclic multi-graph $G = (V, A)$ containing every valid packing pattern for each bin of type t represented as a path from the source $\textsc{s}$ to the target $\textsc{t}_t$, and adding loss arcs connecting each target to the source, the following arc-flow formulation can be used to model the corresponding multiple-choice vector packing problem:

$$\text{minimize} \quad \sum_{t=1}^{q} C_t f_{\textsc{t}_t\textsc{s}}^{0,0} \tag{1}$$

$$\text{subject to} \quad \sum_{(u,v,i,j)\in A:v=k} f_{uv}^{ij} \;-\; \sum_{(u,v,i,j)\in A:u=k} f_{uv}^{ij} = 0 \quad \text{for } k \in V, \tag{2}$$

$$\sum_{(u,v,i,j)\in A:i=k} f_{uv}^{ij} \geq b_k, \qquad k \in \{1,\ldots,m\} \setminus S, \tag{3}$$

$$\sum_{(u,v,i,j)\in A:i=k} f_{uv}^{ij} = b_k, \qquad k \in S, \tag{4}$$

$$f_{uv}^{ij} \leq b_i, \qquad \forall (u,v,i,j) \in A, \tag{5}$$

$$f_{uv}^{ij} \geq 0, \text{ integer}, \qquad \forall (u,v,i,j) \in A, \tag{6}$$

where each arc has four components (u,v,i,j) corresponding to an arc between nodes u and v associated to the incarnation j of item i; arcs $(u,v,0,0)$ are loss arcs; f_{uv}^{ij} is the amount of flow along the arc (u,v,i,j); m is the number of different items; q is the number of bin types; b_i is the demand of items of type i; and $S \subseteq \{1,\ldots,m\}$ is a subset of items whose demands are required to be satisfied exactly for efficiency purposes. For having tighter constraints, one may set $S = \{i = 1,\ldots,m : b_i = 1\}$ (we have done this in our experiments). The only difference in relation to the original arc-flow formulation for VBP is the fact that there is one target node for each bin type instead of just one; on VBP instances this model is exactly the same as the one proposed in Brandão and Pedroso (2016).

In model (1)-(6), the two types of demand constraints and the upper bounds on variable values are mostly useful to take advantage of binary variables and multiple-choice constraints. A simplified version of this general arc-flow formulation that can also be used is the following:

$$\text{minimize} \quad \sum_{t=1}^{q} C_t f_{\textsc{t}_t\textsc{s}}^{0,0} \tag{7}$$

$$\text{subject to} \quad \sum_{(u,v,i,j)\in A:v=k} f_{uv}^{ij} \;-\; \sum_{(u,v,i,j)\in A:u=k} f_{uv}^{ij} = 0, \quad \text{for } k \in V, \tag{8}$$

$$\sum_{(u,v,i,j)\in A:i=k} f_{uv}^{ij} \geq b_k, \qquad k = 1,\ldots,m, \tag{9}$$

$$f_{uv}^{ij} \geq 0, \text{ integer}, \qquad \forall (u,v,i,j) \in A, \tag{10}$$

The following pattern based formulation generalizes Gilmore and Gomory (1961) formulation to the multiple-choice vector packing problem and it is equivalent to model (1)-(6). For bins of type t, let column vectors $a_{kt} = \langle a_{kt}^{ij} \rangle_{(i,j)\in\mathbf{I}}$ represent all feasible packing patterns k; each element a_{kt}^{ij} represents the number of times incarnation j of item i is used in the pattern. Let x_{kt} be a decision variable that designates the number of times pattern k using bins of type t is used. The MVP can be modeled in

terms of these variables as follows:

$$\text{minimize} \qquad \sum_{t=1}^{q} \sum_{k \in K_t} C_t x_{kt} \tag{11}$$

$$\text{subject to} \qquad \sum_{t=1}^{q} \sum_{k \in K_t} \sum_{j \in \mathbf{J}_i} a_{kt}^{ij} x_{kt} \geq b_i, \qquad i = 1, \dots, m, \tag{12}$$

$$x_{kt} \geq 0, \text{ integer}, \qquad t = 1, \dots, q, \ \forall k \in K_t, \tag{13}$$

where every valid packing pattern $k \in K_t$ for bins of type t satisfies:

$$\sum_{(i,j) \in \mathbf{I}} a_{kt}^{ij} w_{ij}^d \leq W_t^d, \qquad d = 1, \dots, p, \tag{14}$$

$$a_{kt}^{ij} \geq 0, \text{ integer}, \qquad (i,j) \in \mathbf{I}. \tag{15}$$

Since there is flow conservation in every node in models (1)-(6) and (7)-(10), the arc-flow solutions are circulations.

Corollary 1 (Flow decomposition for circulations) *Any non-negative feasible circulation flow can be decomposed into the sum of flows around directed cycles.*

Corollary 1 follows directly from the Flow Decomposition Theorem (see, e.g., Ahuja et al. 1993).

Property 1 (Equivalence to the pattern-based model) *For a graph with all valid packing patterns represented as paths from $\textsc{s}$ to $\textsc{t}_t$ for each bin type t, model (7)-(10) is equivalent to the pattern-based model (11)-(13) with the same set of patterns.*

Proof We apply Dantzig-Wolfe decomposition to model (7)-(10) keeping (7) and (9) in the master problem, and (8) and (10) in the subproblem. As the subproblem is a totally unimodular flow model whose solutions can be decomposed into cycles (each including one of feedback arcs associated with a bin type), only valid packing patterns are generated, and we can substitute (8) and (10) by the patterns and obtain a pattern-based model. From this equivalence follows that lower bounds provided by both models are the same when the same set of patterns is considered, and the solution space is exactly the same. □

Property 2 (Equivalence to the pattern-based model) *For a graph with all valid packing patterns represented as paths from $\textsc{s}$ to $\textsc{t}_t$ for each bin type t, model (1)-(6) is equivalent, in terms of lower bound at the root node, to the pattern-based model (11)-(13) with the same set of patterns.*

Proof We apply Dantzig-Wolfe decomposition to model (1)-(6) keeping (1), (3) and (4) in the master problem and (2), (5) and (6) in the subproblem. As the subproblem is a totally unimodular flow model whose solutions can be decomposed into cycles (each including one of feedback arcs associated with a bin type), only valid packing patterns are generated, and we can substitute (2), (5) and (6) by the patterns and obtain a pattern-based model. From this equivalence follows that lower bounds provided by both models are the same when the same set of patterns is considered. The equality constraints (4) and the upper bound on variable values (5) have no effect on the lower bounds, since we are only excluding solutions that satisfy the demand of some items with excess, and for these solutions there are equivalent solutions that do not exceed the demand (recall that every valid packing pattern is represented in the graph). □

3 Arc-flow models and graph compression

For cutting and packing problems, arc-flow models equivalent to pattern-based models can be easily derived from the dynamic programming recursion of the underlying knapsack subproblem (see, e.g., Wolsey 1977). The main challenge is to find a compact representation of the patterns in a reasonable amount of time. In Section 3.1, we show how the arc-flow models can be easily derived from dynamic programming recursions, and in Section 3.2, we show how to obtain smaller arc-flow models using graph compression.

3.1 Deriving arc-flow models from dynamic programming recursions

Definition 1 (Order) *Items are sorted in decreasing order by the sum of normalized weights ($\alpha_{ij} = \sum_{d=1}^{p} w_{ij}^d / \max\{W_t^d : t = 1, \ldots, q\}$), using decreasing lexicographical order in case of a tie.*

The multiple-choice vector packing problem has an underlying knapsack subproblem on the capacity constraints of each bin. Let $\mathbf{I}'$ be the set of items sorted according Definition 1. Let π_i be the profit of item i. For the sake of simplicity, let $w_k = w_{ij}$, $b_k = b_i$, $\pi_k = \pi_i$, and arcs $(u, v, k) = (u, v, i, j)$, for $k = 1, \ldots, |\mathbf{I}'|$ and $(i, j) = \mathbf{I}'_k$. A dynamic programming recursion for the knapsack subproblem can be defined as follows:

$$
D(x, k, c) = \begin{cases}
\min\{C_t : t = 1, \ldots, q, \ x \leq W_t\} & \text{if } k = |\mathbf{I}'| + 1, \\
\min\{D(x + w_k, k, c + 1) - \pi_k, D(x, k + 1, 0)\} & \text{if } c + 1 \leq b_k \text{ and } \exists t \ x + w_k \leq W_t, \\
D(x, k + 1, 0) & \text{otherwise.}
\end{cases}
$$

Note that this dynamic programming recursion could be used to find the most attractive column in a column generation algorithm based on model (11)-(13). At each iteration of the column-generation process, a subproblem is solved and a column (pattern) is introduced in the restricted master problem if its reduced cost is strictly less than zero. Let π_k be the shadow price of the demand constraint associated with item k. The reduced cost of the most attractive pattern is given by $D(\langle 0 \rangle_{d=1,\ldots,p}, 1, 0)$. In our method, instead of using column-generation in an iterative process, we construct a graph containing every valid packing pattern, and this graph can be derived from this dynamic programming recursion.

From this dynamic programming recursion D, we can easily derive an arc-flow model. Consider each dynamic programming state as a node, and each recursive call as an arc. The graph can be obtained as follows. Let $G(x, k, c) = \{((x, k, c), \mathrm{T}_t^s, 0) : t = 1, \ldots, q\}$ if $k = |\mathbf{I}'| + 1$, $G(x, k, c) = \{((x, k, c), (x + w_k, k, c+1), k), ((x, k, c), (x, k+1, 0), 0)\} \cup G(x + w_k, k, c+1) \cup G(x, k+1, 0)$ if $c + 1 \leq b_k$ and $\exists t \ x + w_k \leq W_t$, and $G(x, k, c) = G(x, k + 1, 0)$ otherwise. The source node s is $(\langle 0 \rangle_{d=1,\ldots,p}, 1, 0)$, the set of target nodes is $T = \{\mathrm{T}_t^s : t = 1, \ldots, q\}$ (the base cases of the dynamic programming recursion), the set of arcs is $A = G(\langle 0 \rangle_{d=1,\ldots,p}, 1, 0)$, and the set of vertices is $V = \{u : (u, v, k) \in A\} \cup \{v : (u, v, k) \in A\}$.

The dynamic programming recursion D is equivalent to the following totally unimodular flow problem:

$$
\text{minimize} \quad \sum_{t=1}^{q} \sum_{(u,v,k) \in A : v = \mathrm{T}_t^s} C_t f_{uv}^k \ - \sum_{(u,v,k) \in A : k \neq 0} \pi_k f_{uv}^k \tag{16}
$$

$$
\text{subject to} \quad \sum_{(u,v,k) \in A : u = v'} f_{uv}^k \ - \sum_{(u,v,k) \in A : v = v'} f_{uv}^k = 0, \qquad \text{for } v' \in V \setminus (\{\mathrm{s}\} \cup T), \tag{17}
$$

$$
\sum_{(u,v,k) \in A : u = \mathrm{s}} f_{uv}^k = 1, \tag{18}
$$

$$
f_{uv}^k \geq 0, \qquad\qquad\qquad \forall (u, v, k) \in A. \tag{19}
$$

The dual of this flow problem, which corresponds exactly to the dynamic programming recursion defined above, is the following:

$$
\text{maximize} \quad \theta_{\mathrm{s}}' \tag{20}
$$

$$
\text{subject to} \quad \theta_u' \leq C_t, \qquad \text{for } (u, v, k) \in A, k = 0, \exists t \ v = \mathrm{T}_t^s, \tag{21}
$$

$$
\theta_u' \leq \theta_v', \qquad \text{for } (u, v, k) \in A, k = 0, v \notin T, \tag{22}
$$

$$
\theta_u' \leq \theta_v' - \pi_k, \quad \text{for } (u, v, k) \in A, k \neq 0. \tag{23}
$$

This relationship between dynamic programming recursions for knapsack problems and arc-flow models is one of the results of Wolsey 1977, which leads to a natural method for obtaining arc-flow formulations for cutting and packing problems.

3.2 Graph compression

In the graphs derived from the dynamic programming recursion D, each internal node is connected to at most two other nodes: a node in its level (using the current item), and another in the level above (not using the current item). This graph can be seen as the Step-2 graph of Brandão and Pedroso (2016) as it already divides the graph into levels, one for each item incarnation, and therefore breaks symmetry.

By adding the feedback loss arcs connecting each target node to the source (i.e., $\{(\mathrm{T}_t^s, \mathrm{s}, 0) : t = 1, \ldots, q\}$), the arc-flow model derived directly from the dynamic programming recursion can be used with models (1)-(6) or (7)-(10) to solve the corresponding multiple-choice vector packing problem. However, since there is a constraint for each node and a variable for each arc, the model size may be a problem, and therefore this can only be done for very small instances. Graph compressing is used in order to solve this problem as it usually allows us to obtain reasonably small graphs even when the straightforward approach would lead to models with many millions of variables and constraints.

Graph compression consists of relabelling the graph. In each compression step, a new graph $G' = (V', A')$ is constructed by creating a set of vertices $V' = \{\phi(v) : v \in V\}$ and a set of arcs $A' = \{(\phi(u), \phi(v), k) : (u, v, k) \in A, \phi(u) \neq \phi(v)\}$, where ϕ is the map between the original and new labels. This relabelling procedure must assure that no valid packing pattern is removed, and that no invalid packing pattern is added.

The main compression step is applied to the Step-2 graph (i.e., a graph with level dimension labels which break symmetry). In the Step-3 graph, the longest paths to the target in each dimension are used to relabel the internal nodes ($V \setminus (\{\mathrm{s}\} \cup T)$), dropping the level dimension labels (i.e., k and c) of each node. Let $\langle \varphi^d(u) \rangle_{d=1,\ldots,p}$ be the new label of node u in the Step-3 graph, where

$$\varphi^d(u) = \begin{cases} W_t^d & \text{if } u = \mathrm{T}_t^s \text{ (base case)}, \\ \min_{(u',v,k) \in A : u'=u} \{\varphi^d(v) - w_k^d\} & \text{otherwise.} \end{cases} \tag{24}$$

In the paths from s to T_t^s in the Step-2 graph usually there is slack in some dimension. In this process, we are moving this slack as much as possible to the beginning of the paths.

Finally, in the final compression step, a new graph is constructed once more. In order to try to reduce the graph size even more, we relabel the internal nodes once more using the longest paths from the source in each dimension. Let $\langle \psi^d(v) \rangle_{d=1,\ldots,p}$ be the label of node v in the Step-4 graph, where

$$\psi^d(v) = \begin{cases} 0 & \text{if } v = \mathrm{s} \text{ (base case)}, \\ \max_{(u,v',k) \in A : v'=v} \{\psi^d(u) + w_k^d\} & \text{otherwise.} \end{cases} \tag{25}$$

4   Graph construction and compression algorithm

Algorithm 1 is a generalization of the algorithm proposed in Brandão and Pedroso (2016). It builds the Step-3 graph directly in order to avoid the construction of huge Step-1 and Step-2 graphs that may have many millions of vertices and arcs. We start by sorting the items according the order defined in Defenition 1 in line 3. Algorithm 1 uses dynamic programming to build the Step-3 graph recursively over the structure of the Step-2 graph (without building it). The basic idea for this algorithm comes from the fact that, in the main compression step, the label of any internal node ($\varphi(u) = \langle \min\{\varphi^d(v) - w_{ij}^d : (u',v,i,j) \in A, u'=u\} \rangle_{d=1,\ldots,p}$) only depends on the labels of the two nodes to which it is connected; a node in its level (line 18) and another in the level above (line 14). After directly building the Step-3 graph from the instance's data using this algorithm, we apply the final compression step (line 29) using Algorithm 2, and connect the internal nodes to the targets (line 30) using Algorithm 3. Since parallel arcs associated to the same item type but different incarnations are redundant, all redundant arcs are removed in line 31. In practice, this method allows obtaining arc-flow models even for large benchmark instances quickly.

The dynamic programming states are identified by the space used in each dimension ($\langle x^d \rangle_{d=1,\ldots,p}$), the current item ($k$) and the number of times it has already been used (c). In order to reduce the number

Algorithm 1: Graph construction and compression algorithm

input : $\mathbf{I}$ - set of items; w - item weights; b - demands; q - number of bin types; W - capacity vectors

output: V - set of vertices; A - set of arcs; S - source; T^s - targets

1 **function** buildGraph($\mathbf{I}, w, b, q, W$):

2      $\mathsf{dp}[x, k, c] \leftarrow$ NIL, **for all** x, k, c *// dynamic programming table*

3      $\mathbf{I}' \leftarrow$ reverse(sorted($\mathbf{I}, \text{key} = \lambda(i, j).(\sum_{d=1}^{p} w_{ij}^d / \max\{W_t^d : t = 1, \ldots, q\}, w_{ij})))$ *// sort item incarnations*

4   **function** lift(x, k, c):   *// auxiliary function: lift dp states solving knapsack/longest-path problems in each dimension*

            input : x - used capacity; k - current item; c - number of times the current has been used

5   **function** highestPosition(d, t):

6   **return** $\min \left\{ W_t^d - \sum_{j=k}^{|\mathbf{I}'|} w_{\mathbf{I}'_j}^d y_j : \begin{array}{c} \sum_{j=k}^{|\mathbf{I}'|} w_{\mathbf{I}'_j}^d y_j \leq W_t^d - x^d, \\ y_k \leq b_k - c, \ y_j \leq b_j, \ j = k+1, \ldots, |\mathbf{I}'|, \\ y_j \geq 0, \ \text{integer}, \ j = k, \ldots, |\mathbf{I}'| \end{array} \right\}$

7   **return** $\langle \min\{\text{highestPosition}(d, t) : t = 1, \ldots, q, x \leq W_t\} \rangle_{d=1,\ldots,p}$

8      $(V, A) \leftarrow (\{ \ \}, \{ \ \})$

9   **function** $\varphi(x, k, c)$:

            input : x - used capacity; k - current item; c - number of times the current has been used

10         $x \leftarrow$ lift(x, k, c)   *// lift x in order to reduce the number of dp states*

11   **if** $\mathsf{dp}[x, k, c] \neq$ NIL **then**   *// avoid repeating work*

12   **return** $\mathsf{dp}[x, k, c]$

13         $u \leftarrow \langle \min\{W_t^d : t = 1, \ldots, q, x \leq W_t\} \rangle_{d=1,\ldots,p}$ *// base case of $\varphi(x,k,c)$ if there are no arcs leaving the node*

14   **if** $k < |\mathbf{I}'|$ **then**   *// option 1: do not use the current item (go to the level above)*

15             $up_x \leftarrow \varphi(x, k+1, 0)$

16             $u \leftarrow up_x$ *// value of $\varphi(x,k,c)$ if no more items of the current type are introduced*

17         $(i, j) \leftarrow \mathbf{I}'_k$

18   **if** $c < b_i$ **and** $x + w_{ij} \leq W_t$ **for any** $t = 1, \ldots, q$ **then**   *// option 2: use the current item*

19             $v \leftarrow \varphi(x + w_{ij}, k, c+1)$

20             $u \leftarrow \langle \min(u^d, v^d - w_{ij}^d) \rangle_{d=1,\ldots,p}$ *// update the value of $\varphi(x,k,c)$*

21             $A \leftarrow A \cup \{(u, v, i, j)\}$ *// connect u to the node resulting from option 2*

22             $V \leftarrow V \cup \{u, v\}$

23   **if** $k < |\mathbf{I}'|$ **and** $u \neq up_x$ **then**

24             $A \leftarrow A \cup \{(u, up_x, 0, 0)\}$ *// connect u to the node resulting from option 1*

25             $V \leftarrow V \cup \{up_x\}$

26         $\mathsf{dp}[x, k, c] \leftarrow u$

27   **return** u *// returns $u = \varphi(x, k, c)$*

28      $\mathrm{S} \leftarrow \varphi(x = \langle 0 \rangle_{d=1,\ldots,p}, k = 1, c = 0)$ *// build the graph*

29      $(V, A, \mathrm{S}) \leftarrow$ finalCompression(V, A, S, w)   *// final compression step*

30      $(V, A, \mathrm{S}, \mathrm{T}^s) \leftarrow$ connectTargets(V, A, S, q, W)   *// connect the internal nodes to the targets*

31      $A \leftarrow \{(u, v, i, j) \in A : (u, v, i, j') \notin A, \forall j' < j\}$ *// remove parallel arcs associated to the same item type*

32   **return** $(G = (V, A), \mathrm{S}, \mathrm{T}^s)$

of states, we lift (line 10) each state by solving (using again dynamic programming though this is not explicit in the algorithm) knapsack/longest-path problems in each dimension considering the remaining items (line 5); we try to increase the space used in each dimension to its highest value considering the valid packing patterns for the remaining items. Note that all valid bin sizes must be considered in the lifting procedure.

If there is just one bin type, this algorithm works exactly as the original one. When there are multiple bin types, the main difference are the lift procedure, which needs to take in consideration all valid bin sizes, and the connection of internal nodes to the targets. When there are multiple valid targets for the same node, we need to connect the node to each of them, or take advantage of a transitive reduction

to connect each node to as little targets as possible (as we do in Algorithm 3). For instance, in the variable bin size problem, since there is just one dimension, the transitive reduction allows us to connect each internal node to just one target (i.e., it uses only 1 additional arc per node instead of q additional arcs).

Algorithm 2: Final compression step

---

input : V - set of vertices; A - set of arcs; s - source; w - item weights

output: V - set of vertices; A - set of arcs; s - source

1 function finalCompression(V, A, s, w):

2 $\psi(s) \leftarrow \langle 0 \rangle_{d=1,\ldots,p}$

3   **foreach** $v \in$ sorted($V \setminus \{s\}$) **do**   *// for each vertex in reverse topological order of the transpose graph*

4 $\psi(v) \leftarrow \langle \max\{\psi^d(u) + w^d_{ij} : (u, v', i, j) \in A, \; v' = v\} \rangle_{d=1,\ldots,p}$

5 $s \leftarrow \psi(s)$

6 $V \leftarrow \{\psi(u) : u \in V\}$ *// new set of vertices*

7 $A \leftarrow \{(\psi(u), \psi(v), i, j) : (u, v, i, j) \in A, \; \psi(u) \neq \psi(v)\}$ *// relabel the graph and remove self-loops*

8   **return** $(G = (V, A), s)$

Definition 2 *A bin type t_1 of capacity W_{t_1} dominates a bin type t_2 of capacity W_{t_2}, $(t_1, W_{t_1}) \prec (t_2, W_{t_2})$ for short, if $W_{t_1} = W_{t_2}$ and $t_1 < t_2$, or $W_{t_1} \neq W_{t_2}$ and $W_{t_1} \leq W_{t_2}$.*

Algorithm 3: Connect internal nodes to the targets

---

input : V - set of vertices; A - set of arcs; s - source; q - number of bin types; W - capacity vectors

output: V - set of vertices; A - set of arcs; s - source; T^s - targets

1 function connectTargets(V, A, s, q, W):

2 $T^s \leftarrow \langle T^s_t \rangle_{t=1,\ldots,q}$

3   **foreach** $v \in V \setminus \{s\}$ **do**   *// for each internal node*

4 $\tau \leftarrow \{t : t = 1, \ldots, q, v \leq W_t\}$ *// valid bin types for vertex v*

5   **foreach** $t \in \{1, \ldots, q\}$ **do**

6   **if** $t \in \tau$ **then**

7 $\tau \leftarrow \tau \setminus \{t' \in \tau : (t, W_t) \prec (t', W_{t'})\}$ *// transitive reduction (i.e., remove dominated bin types)*

8 $A \leftarrow A \cup \{(v, T^s_t, 0, 0) : t \in \tau\}$ *// connect v to non-dominated targets*

9   **foreach** $t \in \{1, \ldots, q\}$ **do**   *// for each bin type*

10 $\tau \leftarrow \{t' = 1, \ldots, q : (t, W_t) \prec (t', W_{t'})\}$ *// dominated bin types*

11   **foreach** $t' \in \{1, \ldots, q\}$ **do**

12   **if** $t' \in \tau$ **then**

13 $\tau \leftarrow \tau \setminus \{t'' \in \tau : (t', W_{t'}) \prec (t'', W_{t''})\}$ *// transitive reduction*

14 $A \leftarrow A \cup \{(T^s_t, T^s_{t'}, 0, 0) : t' \in \tau\}$ *// connect T^s_t the targets it directly dominates*

15 $V \leftarrow V \cup \{T^s_t : t = 1, \ldots, q\}$

16 $A \leftarrow A \cup \{(T^s_t, s, 0, 0) : t = 1, \ldots, q\}$ *// add the feedback arcs*

17   **return** $(G = (V, A), s, T^s)$

---

5 Conclusions

In this paper, we presented a graph construction and compression algorithm for multiple-choice vector bin packing problems. This algorithm was introduced in VPSolver 3.0.0 (https://github.com/fdabrandao/vpsolver) as the standard graph construction method. When there is only one bin type and each item has a single incarnation, the new algorithm produces exactly the same graph as the one produced by the original algorithm. When multiple bin types exist, the algorithm produces on a single run a graph containing all the valid packing patterns for all bin types.

References

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows - theory, algorithms and applications.* Prentice-Hall.

Brandão, F. and Pedroso, J. P. (2013). Multiple-choice Vector Bin Packing: Arc-flow Formulation with Graph Compression. Technical Report DCC-2013-13, Faculdade de Ciências da Universidade do Porto, Portugal.

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

Gilmore, P. C. and Gomory, R. E. (1961). A Linear Programming Approach to the Cutting-Stock Problem. *Operations Research*, 9:849–859.

Wolsey, L. A. (1977). Valid inequalities, covering problems and discrete dynamic programs. In P.L. Hammer, E.L. Johnson, B. K. and Nemhauser, G., editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 527 – 538. Elsevier.