Propagating the Bin Packing Constraint using Linear Programming[⋆]

Hadrien Cambazard and Barry O'Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{h.cambazard|b.osullivan}@4c.ucc.ie

Abstract. The state-of-the-art global constraint for bin packing is due to Shaw. We compare two linear continuous relaxations of the bin packing problem, based on the DP-flow and Arc-flow models, with the filtering of the bin packing constraint. Our experiments show that we often obtain significant improvements in runtime. The DP-flow model is a novel formulation of the problem.

1 Introduction

The one-dimensional bin packing problem is ubiquitous in operations research. It is typically defined as follows. Given a set $S = \{s_1, \ldots, s_n\}$ of n indivisible items each of a known positive size s_i, and m bins each of capacity C, can we pack all n items into the m bins such that the sum of sizes of the items in each bin does not exceed C? The one-dimensional bin packing problem is NP-Complete. Amongst the many applications of this problem are timetabling, scheduling, stock cutting, television commercial break scheduling, and container packing.

Our *motivation* comes from a real-world timetabling problem in the Dental School at University College Cork. An interesting characteristic of this problem is that the core challenge relates to solving one-dimensional bin packing problems. This contrasts with many other school timetabling problems which often have challenging list colourings at their core. Our *objective* is to compare two continuous relaxations of the *bin packing* problem with the state-of-the-art filtering algorithm used in the constraint programming community [9]. Continuous relaxations have been developed for many global constraints including *cumulative* [6], *all-different*, *element* and others [5]. We believe that continuous relaxations can be successfully used for the *bin packing* and *knapsack* constraints [9, 10]. This paper presents our initial results in this direction.

2 Linear Programming Formulations for Bin Packing

Numerous linear programming models have been proposed for the bin packing problem [3]. A standard linear model is the following. For each bin $j \in \{1, \ldots, m\}$ we introduce a binary variable y_j which we set to 1 if bin j is used in the packing, and 0

[⋆] This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886).

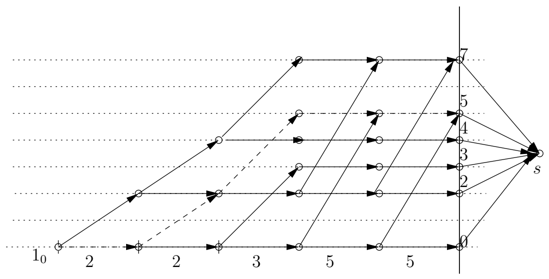


Fig. 1. An example of the graph used by the DP-FLOW model on S = {2,2,3,5,5}, C = 7.

otherwise. For each item $i \in \{1, \ldots, n\}$ and each bin j we introduce a binary variable x_{ij} which we set to 1 if item i is packed into bin j, and 0 otherwise. The full model is:

$$\begin{array}{lll} \textit{minimize} & \sum_{j=1}^{m} y_j & \\ \sum_{j=1}^{m} x_{ij} = 1, & & \forall i \in \{1, \ldots, n\} \\ \sum_{i=1}^{n} s_i \times x_{ij} \leq C y_j, & & \forall j \in \{1, \ldots, m\} \\ x_{ij} \in \{0,1\}, y_i \in \{0,1\} & & \end{array} \quad (1)$$

The lower bound on the number of bins obtained by solving the continuous relaxation of this model is referred to as L_1 in the literature. It has been shown to be equal to $\lceil \sum_{i=1}^{n} s_i / C \rceil$. Many other lower bounds have been designed, amongst which the most widely used is the one due to Martello and Toth [7], referred to as L_2. In the remainder of this section we will present two alternative formulations that give better lower bounds on the number of bins. The first one, referred to the DP-FLOW model, is novel. The second one is known as the ARC-FLOW model [2].

The DP-FLOW Model. We consider the directed graph construction used by Trick [10] to build a propagator for the knapsack constraint using dynamic programming; an example is presented in Figure 1. Consider a layered graph $G(V, A)$ with $n + 1$ layers labelled from 1 to $n + 1$, and a sink node, s. The nodes are labelled i_b where i denotes the layer and b a value between 0 and C. A path from the node of the first layer to a node of the last layer represents a packing, i.e. a set of items assigned to the same bin. More specifically a path using an edge starting at layer i between two nodes i_b and $(i+1)_{b+s_i}$ represents a packing that includes item i, whereas the use of the edge $(i_b, (i + 1)_b)$ excludes item i from the packing. Edges are added between the nodes of the last layer of the graph and the sink node, s. An example of such a graph is shown Figure 1 for the instance $S = \{2, 2, 3, 5, 5\}, C = 7$. The packing $2, 3$ is the path shown in dashed line.

A solution to the bin packing problem corresponds to a minimum flow problem in this graph, with an additional constraint stating that exactly one oblique edge from each layer must be used. We consider a variable $x_{k,l}$ per edge $(k, l) \in A$, 1_0 being the node of the first layer, and the number of bins is represented by $x_{s,1_0}$, i.e. the flow circulating in the graph:

$$\begin{array}{ll} minimise & x_{s,1_0} \\ & \sum_{(j,i_b)\in A} x_{j,i_b} - \sum_{(i_b,k)\in A} x_{i_b,k} = \begin{cases} -x_{s,1_0} & \text{if } i_b = 1_0, \\ 0 & \text{if } i_b \; s.t \; i \in [2, n+1], b \in [0, C] \\ x_{s,1_0} & \text{if } i_b = s \end{cases} \\ & \sum_{(i_b,(i+1)_{b+s_i})\in A} x_{(i_b,(i+1)_{b+s_i})} = 1 \qquad\qquad\qquad i \in [1, n] \\ & x_{k,l} \geq 0, integer \qquad\qquad\qquad\qquad\qquad\qquad \forall (k,l) \in A \end{array}$$

A solution to the bin packing problem can be obtained by decomposing the resulting flow into paths connecting the source node to the nodes of the final layer. This flow decomposition is possible because the graph is acyclic. The number of variables and constraints in this model is $\mathcal{O}(nC)$ since each node in the graph has at most two outgoing edges. Notice that the formulation depends on the ordering of the items used to order the layers of the graph; the size of the graph as well as the strength of the formulation are also affected by this ordering.

The ARC-FLOW Model. Carvalho introduced an elegant ARC-FLOW model for the bin packing problem [2, 3]. His model, which we present below, makes explicit the capacity of the bins, and its size depends on the number of items of different sizes rather than the total number of items.

Consider a graph $G(V, A)$ with $C + 1$ nodes labelled from 0 to C in which there is an edge $(i, j) \in A$ if there exists an item of size $j - i$ in S. Additional edges $(i, i + 1)$ are added between consecutive pair of nodes. An example of such a graph is shown in Figure 2 for $S = \{2, 2, 3\}$ with $C = 5$. Any path in this graph corresponds to a packing of a single bin. For example, the path shown in dotted lines in Figure 2 corresponds to a packing of two items of size 2, leaving the remaining capacity of the bin unused (the last edge is a loss edge). More formally, a packing for a single bin corresponds to a flow of one unit between vertices 0 and C. A larger flow corresponds to the same packing into multiple bins.

Such a formulation has many symmetries since the same solution can be encoded by many different flows. Some reduction rules were given by Carvalho that help reduce such symmetries [2]. The graph presented in Figure 3 is a simplied graph for the same example as the one used for the DP-FLOW model. Firstly one can notice that the packings are ordered by decreasing value of the sizes of the items. Secondly, the loss edges before Node 2, which is the smallest item size, have been removed as well. Finally, the number of consecutive edges of a given size is bounded by the number of items of

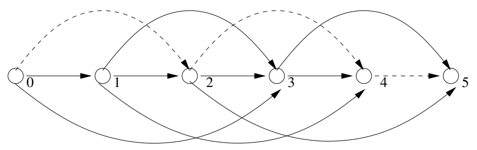


Fig. 2. An example of the graph underlying the ARC-FLOW model for $S = \{2, 2, 3\}$ and $C = 5$. The packing $2, 2$ is shown with dotted lines.

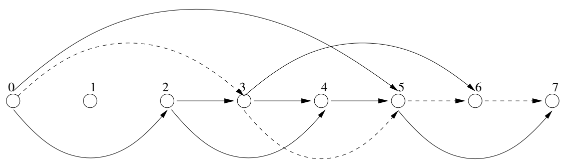


Fig. 3. An example of the graph underlying the ARC-FLOW model for S = {2,2,3,5,5}, C = 7.

this size. This is why no edges of size 2 are outgoing from Node 4 as this would not correspond to any valid packing. However, all the symetries have not been eliminated.

The bin packing problem can be formulated as a minimum flow between vertex 0 and vertex C with constraints enforcing that the number of edges of a given length used by the flow must be greater than or equal to the number of items of the corresponding size. Variables x_{ij} are associated with the edges (i, j). x_{C0} denotes the flow variable corresponding to the number of bins used. We will denote by $S^{'} = \{s^{'}_{1}, \ldots, s^{'}_{n'}\}$ the set of different item sizes, and b_i the number of items of size $s^{'}_{i}$. $n^{'}$ is the number of items of different sizes. The model is as follows:

$$\begin{array}{ll}
minimise & x_{C0} \\
\sum_{(i,j)\in A} x_{ij} - \sum_{(j,k)\in A} x_{jk} = \begin{cases} -x_{C0} & \text{if } j = 0, \\ 0 & \text{if } j = 1, 2, \ldots, C-1, \\ x_{C0} & \text{if } j = C \end{cases} & \\
\sum_{(k,k+s^{'}_{i})\in A} x_{k,k+s^{'}_{i}} \geq b_i & i = 1, 2, \ldots, n^{'} \\
x_{i,j} \geq 0, integer & \forall (i,j) \in A
\end{array}$$

A solution can be obtained again by decomposing the flow. The number of variables in this model is $\mathcal{O}(n^{'}C)$. However there are only C flow conservation constraints, as opposed to the nC of our model, which makes this formulation clearly more compact.

3 Dealing with Partial Assignments

The bounds we have presented can be applied during search by transforming a partial assignment into a reduced bin packing problem. Once items are assigned to bins we have a bin packing problem in which some item sizes are missing, since they have already been assigned, and not all bins have the same remaining capacity. Both previous formulations can be modified to handle these cases.

For the DP-FLOW model we can simply add capacities to the edges between the last layer of the graph and the sink node. These capacities express the number of bins that have enough capacity to accommodate the corresponding size. For example, if we dispose of three bins of capacity 10 and an item of size 2 has been assigned to each of the first two bins, then sizes of value 9 and 10 are given a capacity of 1. Additionally, the flows in the oblique edges corresponding to taking items already assigned are enforced to zero. For the ARC-FLOW model a back edge, adding to the overall flow to minimize, can be added from each node corresponding to an available capacity in the reduced bin

Table 1. Comparing the quality and time of various lower bounds on the B1 benchmark.

	L_1	L_2	DP-FLOW	ARC-FLOW	ARC-FLOW+red
sum	74650.49	77945.76	78114.48	78099.66	78113.4
avg time (in s)	0	0	2.96	0.07	0.02

packing problem. In the previous example an edge would leave Node 8 as well as 10 to go back to Node 0. The b_i values of the linear model are also updated accordingly to reflect the remaining items available.

It is possible to propagate with the LP lower bounds using a similar approach to that adopted by Shaw [9]; that is, we simply commit items to bins, compute the corresponding reduced problem and check whether the bound raises a contradiction or not, in which case the item can be pruned from the corresponding bin. However, that approach is suitable for very fast filtering rules only, but otherwise leads to significant overheads.

4 Experimental Results

We conducted a series of experiments on a single thread on a Dual Quad Core Xeon CPU, 2.66GHz with 12MB of L2 cache per processor and 16GB of RAM overall, running Linux 2.6.25 x64. We put a 3GB limit on memory. CPLEX 12 was used for all the linear models. We used two sets of publicly available instances as benchmarks. The Falkenauer benchmark [4] comprises two classes of instances U and T with four sets of 20 instances in each class containing 120 to 1000 items. Class U comprises item sizes uniformly distributed in $[20, 100]$ to be packed into bins of size 150. Class T consists of triplets of items from $[25, 50]$ to be packed into bins of size 100. Four sets are in class T and instances were generated so that there is no slack. The second benchmark suite used are the B1 and B2 sets studied in [8], made of 720 and 480 instances, respectively. The number of items in these sets vary from 50 to 500; the capacity can reach 1000 in the $B2$ set. We used the first 350 instances of B2. When quoting a benchmark we will use either U, T or B as prefixes followed by the number of items to be packed, unless it is otherwise obvious.

Experiment 1: Comparison of the Lower Bounds on the Number of Bins. We compared four lower bounds: (a) L_1 is the continuous lower bound, (b) L_2 is the well known bound due to Martello and Toth [7], (c) the linear relaxation of the DP-FLOW model where the ordering of the layers in the graph is done by non-decreasing item size, (d) the linear relaxation of the ARC-FLOW model without the simplifications of the graph, and (e) with the reductions. The lower bounds obtained on the Falkenauer benchmark at the root node are not interesting, and mostly equal to the continuous bound L_1 with few exceptions. The second benchmark, B1, exhibits more variety and we report in Table 1 the sum of the lower bounds found on all the instances of the B1 set by the five lower bounds. The linear relaxation of DP-FLOW is the strongest but does not improve ARC-FLOW+red significantly and is also significantly more expensive to compute.

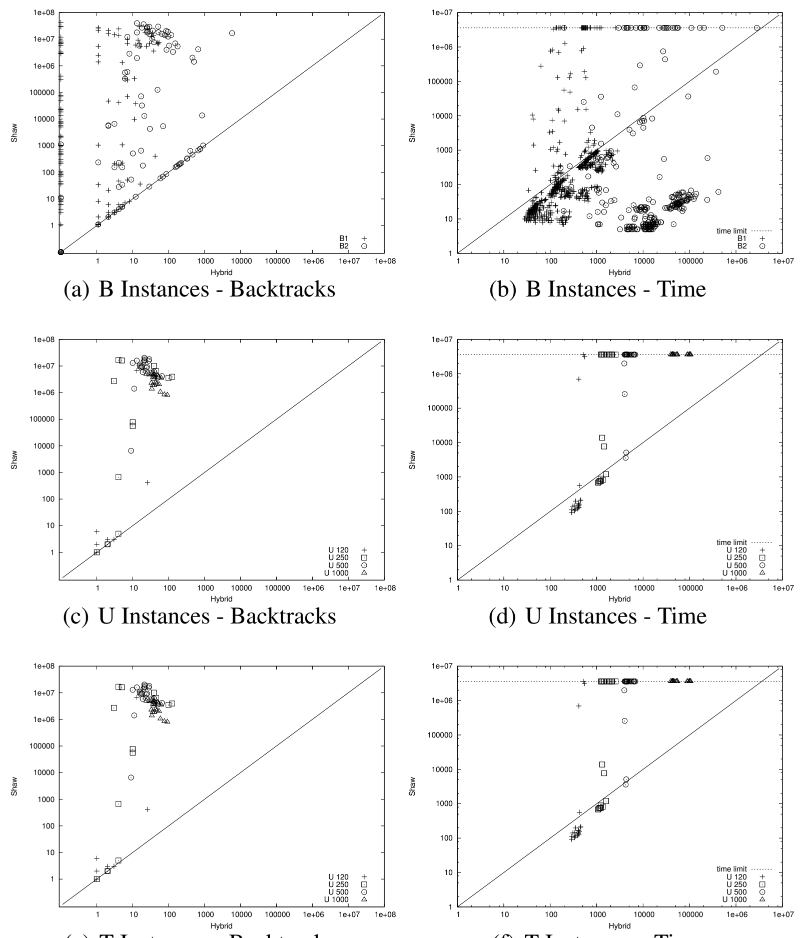
Experiment 2: Comparison with the Bin Packing Constraint. We embedded the ARC-FLOW+red bound within a bin packing global constraint to evaluate the strength of the pruning we would get compared to that obtained using the state-of-the-art global constraint designed by Shaw [9]. We compared these three methods by optimally solving the bin packing problem: “CP Shaw” denotes the constraint described in [9]; “CP Arc-flow” refers to a constraint that simply solves the linear relaxation of the ARC-FLOW model with reductions and uses this lower bound to detect contradiction; “CP Shaw+Arc-flow” first applies the filtering of Shaw’s bin packing constraint and then computes the lower bound of the ARC-FLOW model.

Table 2. Comparison between different variants of the bin packing constraint.

benchmark			u120	u250	u500	u1000	t60	t120	t249	t501	B1	B2
CP	Backt	Med	0.00	3.13M	8.68M	2.95M	147.00	5518.00	1.79M	2.41M	0.00	0.00
Sh		Avg	1.41M	4.83M	9.10M	3.01M	216.15	1.54M	2.81M	2.37M	1.15M	3.21M
	Time(s)	Med	0.15	3600.00	3600.07	3600.09	0.07	3.06	3600.07	3600.09	0.34	0.06
		Avg	371.61	1981.41	2992.87	3600.10	0.13	741.78	3265.10	3600.09	497.11	726.09
		StDev	1038.75	1835.99	1302.88	0.06	0.14	1467.85	1033.12	0.06	1224.44	1439.92
	NS		19	9	4	0	20	16	2	0	625	280
CP	Backt	Med	15.00	76.00	175.00	285.00	313.00	1093.00	3604.00	8535.50	0.00	0.00
AC		Avg	25.75	87.80	177.85	268.45	318.20	1100.80	14676.05	16240.85	15.76	132.40
	Time(s)	Med	0.29	0.71	0.96	1.45	13.18	58.10	142.51	208.05	0.21	29.26
		Avg	0.34	0.73	1.02	1.45	13.34	64.92	379.02	408.70	0.54	246.49
		StDev	0.12	0.11	0.15	0.08	4.37	37.95	788.56	757.92	0.96	631.00
	NS		20	20	20	20	20	20	19	19	720	343
CP	Backt	Med	0.00	7.50	21.00	40.50	7.00	33.00	173.50	645.00	0.00	0.00
Sh+AC		Avg	3.95	23.10	24.15	46.10	7.50	39.65	4420.00	3981.50	3.65	68.29
	Time(s)	Med	0.40	1.38	4.44	48.86	3.73	5.65	29.10	76.75	0.48	19.56
		Avg	0.39	1.53	4.86	61.84	3.72	17.43	221.79	272.69	2.28	150.39
		StDev	0.07	0.40	0.93	25.06	1.98	15.10	796.55	785.83	3.92	421.52
	NS		20	20	20	20	20	20	19	19	720	349

We use similar settings to those in [9]. The standard search heuristic *decreasing best fit* is used, packing items in non-increasing size into the first bin with the least possible space sufficient for the item. On backtracking a bin is removed from the domain of an item and two symmetry breaking rules are used: the bin is also removed from the domain of all equivalent items (items of the same size), and all equivalent bins (bins with same load) are also pruned from the domains of these items. Two dominance rules are also added. The first is applied before creating a new choice point: if all the bins are equivalent for any item it is assigned to the first available one. The second ensures that if an item can fit exactly in the remaining space of a bin it is assigned to this bin.

Our algorithms are implemented with Choco and CPLEX. Shaw’s bin packing constraint is available in Choco, which we used as a baseline. It differs from Shaw’s implementation by using several dual feasible functions [1] that subsume the L_2 bound of Martello and Toth. The results presented in [9] are detailed for the T60 and U120 categories where all instances are solved, but $U120_{19}$ requires 15 hours. We observed a similar behavior: $U120_{19}$ was the only instance not solved within the one hour time limit allowed for the Choco implementation of the bin packing constraint. Table 2 reports the comparison giving for each category the median and average number of backtracks, time (in seconds), as well as the standard deviation in time and the number of instances solved to optimality within the time limit (NS). Clearly the approach presented here significantly improves over Shaw’s bin packing constraint. In Figure 4 we present an instance by instance comparison for each benchmark suite using both Shaw’s global constraint and our hybrid approach. Our approach tends to have a less variation



(a) B Instances - Backtracks (b) B Instances - Time

(c) U Instances - Backtracks (d) U Instances - Time

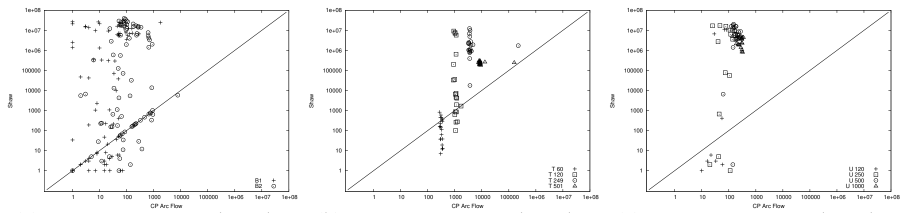
(e) T Instances - Backtracks (f) T Instances - Time

Fig. 4. A detailed comparison of our approach (x-axis) against the bin packing constraint of Shaw on the y-axis - time and backtracks are presented.

in the effort required to solve a bin packing problem. Finally, in Figure 5 we compare the reasoning capabilities of the ARC-FLOW relaxation against that of Shaw's global constraint. A point below the diagonal means that Shaw has fewer backtracks than the ARC-FLOW relaxation. The ARC-FLOW alone is usually better than Shaw, by capturing most of the benefits of more sophisticated filtering while achieving orders of magnitude improvements in time.

5 Conclusion

We have presented a novel direction for handling bin packing constraints in constraint programming. Our approach can give significant improvements in running time. The



(a) B Instances - Backtracks (b) T Instances - Backtracks (c) U Instances - Backtracks

Fig. 5. Comparison of the reasoning capabilities of Shaw's global constraint against ours.

DP-FLOW and ARC-FLOW models remain to be compared theoretically. A deeper study of the DP-FLOW model focusing on the impact of the ordering of the layers and graph reduction criteria is still to be carried out, although the ARC-FLOW model appears to be much more promising. Furthermore we believe that the graph underlying the ARC-FLOW model scales to much larger problems than the one used by the DP-FLOW model. This immediately suggests that we could apply the same idea to knapsack constraints. The resulting formulation would not be able to provide GAC for the knapsack constraint as opposed to [10] but should allow a very strong propagation in practice while scaling to much bigger knapsacks.

References

1. F. Clautiaux, C. Alves, and J. M. Valério de Carvalho. A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, 2008.
2. José M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86(0):629–659, 1999.
3. José M. Valério de Carvalho. Lp models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002.
4. Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.
5. John N. Hooker. *Integrated Methods for Optimization (International Series in Operations Research & Management Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
6. John N. Hooker and Hong Yan. A relaxation of the cumulative constraint. In *CP*, pages 686–690, 2002.
7. S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Appl. Math.*, 28(1):59–70, 1990.
8. Armin Scholl, Robert Klein, and Christian Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research*, 24(7):627 – 645, 1997.
9. Paul Shaw. A constraint for bin packing. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer, 2004.
10. Michael A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals OR*, 118(1-4):73–84, 2003.