Failure detection for the Bin-Packing Constraint

J. Dupuis[1], P. Schaus[2], and Y. Deville[1]

[1]Université catholique de Louvain, Belgium,
{julien.dupuis,yves.deville}@uclouvain.be
[2]Dynadec Europe, Belgium, pschaus@dynadec.com

June 26, 2009

Abstract

In addition to a filtering algorithm, the Pack constraint introduced by Shaw uses a failure detection algorithm. This test is based on a reduction of the partial solution to a standard bin-packing problem and the computation of a bin-packing lower bound (BPLB) on the reduced problem. A first possible improvement on Shaw's test is to use a stronger BPLB. In particular, Labbé's lower bound was recently proved to dominate Martello's lower bound used by Shaw. A second possible improvement is to use a reduction different from the one introduced by Shaw. We propose two new reduction algorithms and prove that one of them theoretically dominates the others. All the proposed improvements on the failure test are evaluated using the COMET System.

1 Introduction

A standard bin-packing problem (SBP) consists in finding the minimum number of bins necessary to pack a set of items so that the total size of the items in one bin does not exceed the bin capacity. The bin capacity is common for all the bins.

This problem can be solved in CP by introducing one placement variable x_i for each item and one load variable l_j for each bin. The $\texttt{Pack}([x_1,\ldots,x_n],[w_1,\ldots,w_n],[l_1,\ldots,l_m])$ constraint introduced by Shaw [8] links the placement variables $x_1,\ldots,x_n$ of n items having weights $w_1,\ldots,w_n$ with the load variables of m bins $l_1,\ldots,l_m$. More precisely the constraint ensures that $\forall j \in [1..m] : l_j = \sum_{i=1}^{n}(x_i = j)\cdot w_i$ where $x_i = j$ is reified to 1 if the equality holds and zero otherwise.

The Pack constraint was successfully used in several applications such as Assembly Line Balancing Problems [5], Steel Mill Slab Design Problems [2] or Nurse Rostering Problems [6].

In addition to the decomposition constraints $\forall j \in [1..m] : l_j = \sum_{i=1}^{n}(x_i = j)\cdot w_i$ and the redundant constraint $\sum_{i=1}^{n} w_i = \sum_{j=1}^{n} l_j$, Shaw introduced:

1. a filtering algorithm based on a knapsack reasoning inside each bin, and

2. a failure detection algorithm based on a reduction of the partial solution to a standard bin-packing problem.

This work focuses on improvements of the failure detection algorithm.

Outline Section 2 recalls the reduction of the partial solution to a bin-packing problem introduced by Shaw and two new reductions are introduced. Section 3 compares theoretically the strengths of the reductions. Section 4 presents the bin-packing lower bounds applied on the reduced problems. Section 5 evaluates the different variants of the failure detection on generated instances and on Scholl's SALBP-1 benchmark [7].

2 Reductions to bin packing problems

In the second part of his paper [8], Paul Shaw gives a fast failure detection procedure for the `Pack` constraint using bin-packing lower bound (BPLB). His idea is to reduce the current partial solution (i.e. considering the already assigned items in the current domains) of the `Pack` constraint to a standard bin packing problem. Then a fail is detected if the BPLB is larger than the number of available bins m.

We propose two new reductions of a partial solution to a bin packing problem. The first one can in some cases dominate Shaw's reduction and the second one dominates the other two theoretically.

We abbreviate as FDSBP the problem of detecting a failure by reduction of a partial solution of the `Pack` constraint to a SBP.

Shaw's reduction: FDSPB0 Paul Shaw's reduction (FDSBP0) consists in creating a standard bin packing (SBP) problem with the following characteristics. The bin capacity is the largest upper bound of the load variables *i.e.* $c = \max_{j \in [1..m]}(l_j^{\max})$. All items that are not packed in the constraint are part of the items of the reduced problem. Furthermore, for each bin, a virtual item is added to the reduced problem to reflect (1) the upper bound dissimilarities of the load variables and (2) the already packed items. More precisely, the size of the virtual item for a bin j is $(c - l_j^{\max} + \sum_{\{i|x_i=j\}} w_i)$ that is the bin capacity c reduced by the actual capacity of the bin in the constraint plus the total size of the already packed items into this bin. An example is shown in Figure 1.

FDSBPMin We introduce FDSBPMin that is obtained from FDSBP0 by reducing the capacity of the bins and the size of all the virtual items by the size of the smallest virtual item. The virtual items have a size of $(c - l_j^{\max} + \sum_{\{i|x_i=j\}} w_i - min_k(c - l_k^{\max} + \sum_{\{i|x_i=k\}} w_i))$. This reduction is illustrated in Figure 1(c).

FDSBPMax We propose FDSBPMax that consists in increasing the capacity and the size of the virtual items by a same amount, so that it is guaranteed that, when distributing the items with a bin-packing algorithm, each virtual item will occupy a different bin. In order to achieve this, each virtual item's size must be larger than half the bin capacity.

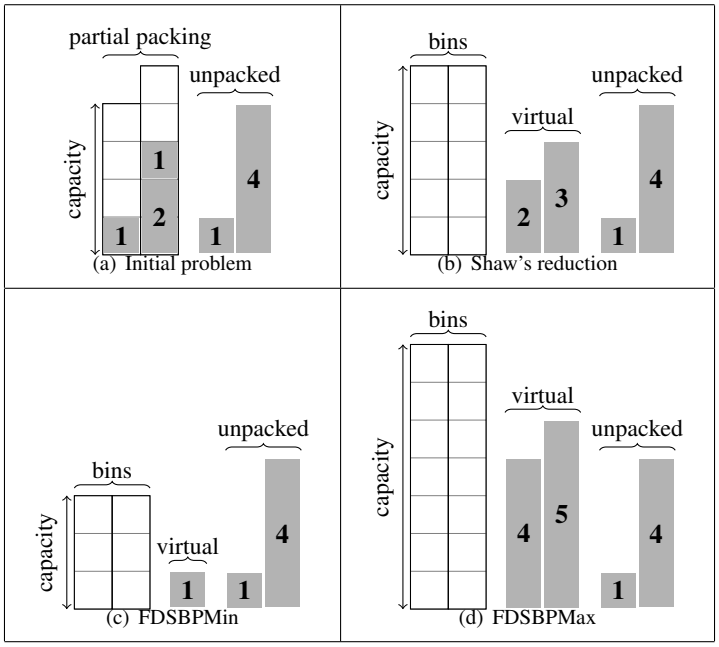

Figure 1: Example of the three reductions for the bin-packing problem

In FDSBP0, let p be the size of the smallest virtual item, and c the capacity of the bins. The size of the virtual items and the capacity must be increased by $(c - 2p + 1)$. The smallest virtual item will have a size of $s = (c - p - 1)$ and the capacity of the bins will be $(2c - 2p + 1) = 2s - 1$. As one can observe, the smallest virtual item is larger than the half of the capacity. If $c = 2p - 1$, this reduction is equivalent to Shaw's reduction. Note that if $c < 2p + 1$, the capacity and the virtual items will be reduced.

The virtual items have a size of $(2c - 2p + 1 - l_j^{\max} + \sum_{\{i|x_i=j\}} w_i)$.

This reduction is illustrated in Figure 1(d).

Generic reduction: FDSBP All these reductions are particular cases of a generic reduction (FDSBP), which, based on FDSBP0, consist in adding a (positive or negative) delta (δ) to the capacity and to all the virtual items' sizes.

For FDSBP0, $\delta = 0$. For FDSBPMin, δ is the minimum possible value that keeps all sizes positive. A smaller δ would create an inconsistency, as the smallest virtual item would have a negative size. $\delta_{FDSBPMin}$ is always negative or equal to zero. For FDSBPMax, δ has the smallest value that guarantees that items cannot pile up. Note that in some cases, $\delta_{FDSBPMin}$ or $\delta_{FDSBPMax}$ can be zero. Also note that δ_{FDSBP0} can be larger than the others.

3 Theoretical comparison of the three reductions

Dominate Let A and B be two instances of standard bin-packing. We say that A *dominates* B ($A >> B$) if the number of bins required in A is larger that the number of bins required in B.

Theorem 3.1 *FDSBP is a relaxation of the problem of testing the consistency of the* `Pack` *constraint.*

Proof If a partial solution of the `Pack` constraint can be extended to a solution with every item placed, then FDSBP has also a solution: if each virtual item is placed in its initial bin, then the free space of each bin is equal to its free space in the partial solution, and so all the unplaced items can be placed in the same bin as in the extended solution from the partial assignment.

Theorem 3.2 *FDSBP0 and FDSBPMin are not comparable*

Proof Figure 2 shows an instance where FDSBP0 has a solution and FDSBPMin does not. Figure 3 shows an instance where FDSBPMin has a solution and FDSBP0 does not.

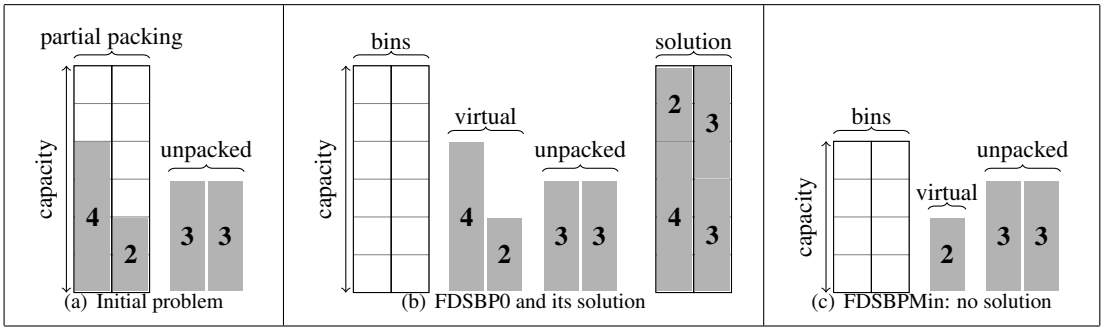

Figure 2: Bin-packing instance where FDSBP0 cannot detect the failure

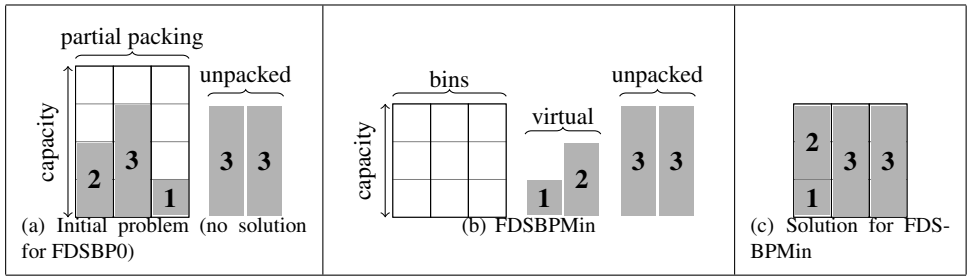

Figure 3: Bin-packing instance where FDSBPMin cannot detect the failure

Theorem 3.3 *FDSBPMax is equivalent to testing the consistency of the* `Pack` *constraint*

Proof By Theorem 3.1, FDSBPMax is a relaxation of the partial solution of the BPP. There remains to show that if there is a solution for FDSBPMax (no failure detected) then the partial solution can be extended to a complete solution of the `Pack` constraint. Let's call v the bin from which the virtual item v is from. It is guaranteed by the size

of the virtual items that they will each be placed in a different bin b_v. The remaining space in each bin b_v corresponds to the free space in bin v in the original problem. The extended solution of the `Pack` constraint is obtained by packing in v all items packed in b_v.

Corollary 3.4 *FDSBPMax dominates FDSBP0 and FDSBPMin.*

4 Bin-Packing Lower Bounds

The failure test of Shaw [8] uses the bin-packing lower bound $\mathcal{L}_2$ of Martello and Toth [4] that can be computed in linear time. Recently the lower bound $\mathcal{L}_3$ of Labbé [3] has been proved to be always larger than or equal to $\mathcal{L}_2$ and to benefit from a better worst case asymptotic performance ratio ($3/4$ for $\mathcal{L}_3$ [1] and $2/3$ for $\mathcal{L}_2$ [4]) for a same linear computation time. We recall the definitions of these two lower bounds.

Let us denote by $N = \{1, \ldots, n'\}$ the set of items to be packed with their weights $w_1, \ldots, w_{n'}$, and by c the bin capacity. We formulate the lower bounds $\mathcal{L}_2$ and $\mathcal{L}_3$ following the notations from [1] with $W(a, b) = \{i \in N | a < w_i \leq b\}$ and $\overline{W}(a, b) = \{i \in N | a \leq w_i \leq b\}$:

$$
\begin{aligned}
&\mathcal{L}_2 = \max_{0 \leq \nu \leq c/2} \{\mathcal{L}_2(\nu)\} \\
&\mathcal{L}_2(\nu) = |W(c/2, c)| + l(\nu) \\
&l(\nu) = \max\left(0, \left\lceil \frac{\sum_{i \in \overline{W}(\nu, c-\nu)} w_i - |W(c/2, c-\nu)| \cdot c}{c} \right\rceil\right) \\
&\mathcal{L}_3 = \max_{0 \leq v \leq c/3} \{\mathcal{L}_3(\nu)\} \\
&\mathcal{L}_3(\nu) = |W(c/2, c)| + \lceil H/2 \rceil + p(\nu) \\
&p(\nu) = \max\left(0, \left\lceil \frac{\sum_{i \in \overline{W}(\nu, c-\nu)} w_i - (|W(c/2, c-\nu)| + \lceil H/2 \rceil) \cdot c}{c} \right\rceil\right)
\end{aligned}
$$

Where H denotes the set of items having their weight in $]c/3, c/2]$ that cannot be matched with items having their weight in $]c/2, 2c/3]$.

We give some intuitions on each terms of these bounds:

- $|W(c/2, c)|$: For both bounds each items from $W(c/2, c)$ must be placed in a different bin since two items from this set cannot fit together in a bin.
- $l(\nu)$: All the items with size larger or equal to ν cannot be placed with any items from $W(c-\nu, c)$ (items smaller than ν are not considered). With $|W(c/2, c)|$, we already packed each items of $W(c/2, c-\nu)$ in separate bins. The idea is to fill preemptively and completely these bins with the items of $\overline{W}(\nu, c-\nu)$. The left over $(\sum_{i \in \overline{W}(\nu, c-\nu)} w_i - |W(c/2, c-\nu)| \cdot c)$ is used to fill (preemptively) new bin(s). Since different values for ν can lead to different $l(\nu)$, the maximum over ν is taken as the contributing term in the bound. The explanation of the term $p(\nu)$ in $\mathcal{L}_3$ is similar to the one of $l(\nu)$.

- $\lceil H/2 \rceil$ in the computation of $\mathcal{L}_3$: a bin cannot contain more than two items from the set $W(c/3, c/2)$ and of course no more than two items from the set $W(c/3, c)$. We try to pair as many items as possible from the set $W(c/3, c/2)$ with the items from $W(c/2, c)$ already placed in separate bins. The items from $W(c/3, c/2)$ that could not be matched are paired together in different bins. This is the origin of the term $\lceil H/2 \rceil$ in the bound $\mathcal{L}_3$.

Both bounds can be computed in linear time when the item sizes are sorted. Note that only the values for ν corresponding to a size in the item set need to be tested.

It has been shown in [1] that $\mathcal{L}_3 \geq \mathcal{L}_2$. Next example illustrates a case where $\mathcal{L}_3$ dominates $\mathcal{L}_2$:

Example Let $c = 10$ and the weight vector be $[4, 4, 4, 4, 4]$. For both bounds $W(c/2, c) = \phi$. For the $\mathcal{L}_2$ bound : $\mathcal{L}_2(0) = \mathcal{L}_2(4) = \lceil (20 - 0)/10 \rceil = 2$. Hence $\mathcal{L}_2 = 0 + 2 = 2$. For bound $\mathcal{L}_3$, $H = \{4, 4, 4, 4, 4\}$. The only contributing term in $\mathcal{L}_3$ is $\lceil H/2 \rceil = 3$.

5 Experimental comparison

5.1 Experimental comparison of the three reductions

Throughout these experiments, we are using the lower bound $\mathcal{L}_3$.

Although in theory, FDSBPMax always outperforms FDSBP0 and FDSBPMin, the practical results are less systematic. This is because $\mathcal{L}_3$ has no guarantee to be monotonic which means that an SBP instance requiring a larger number of bins than a second instance can have a lower bound smaller than the second one.

In fact, $\mathcal{L}_3$ is more adapted to instances where most item sizes are larger than the third of the capacity. FDSBPMax increases the capacity, making unpacked items proportionally smaller.

There is no absolute rule telling which of the three reductions will provide the best results, as for each of them, there are instances for which it outperforms the other two. Table 1 presents three instances for which each reduction combined with $\mathcal{L}_3$ is the only one to detect a failure.

Capacity	Partial packing	Unplaced items	Failure detected		
			FDSBP0	FDSBPMin	FDSBPMax
5	[2,2,1]	[4,2,2,2]	YES	NO	NO
6	[3,3]	[2,2,2]	NO	YES	NO
6	[4,4,2]	[3,3]	NO	NO	YES

Table 1: Instances for which only one reduction gives a failure

The difference of results between the reductions is unpredictable and depends on the problems, because two factors influence the result. The larger δ is, the smaller some items become with respect to the capacity of the bins. But the smaller δ is, the more virtual items will pile up in the algorithm.

Table 2 shows the performance of the failure detection by each one of the reductions. Additional reductions have been experimented, with δ being respectively 25%, 50% and

75% on the way between $\delta_{FDSBPMin}$ and $\delta_{FDSBPMax}$. These results where obtained by generating more than 1,000 random instances and computing $\mathcal{L}_3$ on each of their reductions.

Here is how the instances were produced:

1 Number of bins, number of items and capacity each randomly chosen between 30 and 50. Bins already filled up to 1..capacity. Size of the items randomly chosen between 1 and the capacity.

2 50 bins. Capacity = 100. Number of items is 100 or 200. Size with normal distribution ($\mu = 5000/n$, $\sigma \in \{3n, 2n, n, n/2, n/3\}$ where n is the number of items). Among these, percentage of items already placed $\in \{10\%, 20\%, 30\%, 40\%, 50\%\}$.

3 Idem, but number of placed items is 90% or 95%.

Experiment	Number of failures detected (%)					
	FDSBPMin	FDSBP25	FDSBP50	FDSBP75	FDSBPMax	FDSBP0
1	74.16	78.87	86.40	89.53	99.58	74.79
2	99.93	86.75	87.03	87.8	87.15	99.93
3	80.64	86.55	93.37	97.75	99.39	98.52

Table 2: Comparison of the number of failures found with different reductions

This reveals that some problems are more adapted to FDSBP0, while some are more adapted to FDSBPMax.

5.2 Comparison on real instances

For the analysis to be more relevant, we compared the behavior of the three proposed reductions on real instances. A CP algorithm was run over Scholl's SALBP-1 benchmark, and at every change in the domains of the variables, the current partial solution was extracted. We randomly selected 30,000 extracted instances.

The three reductions and $\mathcal{L}_3$ were applied on these selected instances. Table 3 gives a summary of the results.

only FDSBP0	1	0.03%
only FDSBPMin	140	4.06%
only FDSBPMax	51	1.48%
all but FDSBP0	22	0.64%
all but FDSBPMin	345	10.00%
all but FDSBPMax	511	14.81%
all	2381	68.99%
total FDSBP0	3238	93.83%
total FDSBPMin	3054	88.49%
total FDSBPMax	2799	81.11%

Table 3: Failure detection on benchmark problems using the reductions

For 3451 of these instances, a failure was detected with at least one of the reductions.

These results show that FDSBP0 detects the larger number of failures. But it is also the one with most of the detected failures also detected with another reduction. The conjunction of FDSBPMin and FDSBPMax (with a total of 3450 failure detections) is better than FDSBP0 alone.

As the failure detection with any of these reductions is computed in linear time, we think it is a good idea to consider all of them, or at least FDSBPMin and FDSBPMax together. If only one reduction is to be applied, FDSBP0 (Shaw's reduction) remains the best choice.

5.3 Comparing $\mathcal{L}_2$ and $\mathcal{L}_3$

As an improvement to Shaw's failure detection algorithm, we are using Labbé's lower bound $\mathcal{L}_3$ instead of $\mathcal{L}_2$. We executed both lower bound algorithms on the instances presented in section 5.2.

Table 4 presents the comparison between $\mathcal{L}_2$ and $\mathcal{L}_3$.

Reduction	$\mathcal{L}_3$	$\mathcal{L}_2$	Total
FDSBP0	3238 (100%)	2817 (99.6%)	3238
FDSBPMin	2799 (100%)	2165 (77.3%)	2799
FDSBPMax	3054 (100%)	2817 (99.6%)	3054

Table 4: Failure detection on benchmark problems using the reductions

The difference between $\mathcal{L}_2$ and $\mathcal{L}_3$ is only substantial for FDSBPMin. This can be expected, as a difference occurs when many items have a size between the third and the half of the capacity. With FDSBPMin, the item sizes are larger proportionally to the capacity, so that more tasks do not have a negligible size.

6 Conclusion

We presented two new reductions of a partial solution of the `Pack` constraint to a standard bin-packing problem. Through a CP search, these reductions are submitted to a bin-packing lower bound algorithm in order to detect failures of the `Pack` constraint as suggested by Shaw in [8].

We proved that our second reduction (FDSBPMax) provides a better failure detection than the others, assuming a perfect lower-bound algorithm.

Though Shaw's reduction (FDSBP0) is the most powerful for failure detection in benchmark cases, we proved that it does not dominate the others in all cases, and that considering different reductions can improve the pruning. Applying a lower bound algorithm on these reductions is done in linear time so our conclusion is that it is worth considering all three reductions in an actual CP search.

References

[1] Jean-Marie Bourjolly and Vianney Rebetez. An analysis of lower bound procedures for the bin packing problem. *Comput. Oper. Res.*, 32(3):395–405, 2005.

[2] P. Van Hentenryck and L. Michel. The steel mill slab design problem revisited. *CP'AI'OR-08, Paris, France*, 5015:377–381, May 2008.

[3] Martine Labbé, Gilbert Laporte, and Hélène Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622, 1991.

[4] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Appl. Math.*, 28(1):59–70, 1990.

[5] P. Schaus and Y. Deville. A global constraint for bin-packing with precedences: Application to the assembly line balancing problem. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 369–374, Chicago, Illinois, USA, July 2008. AAAI Press.

[6] P. Schaus, P. Van Hentenryck, and J.C. Régin. Scalable load balancing in nurse to patient assignment problems. In *6th International Conference Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, Lecture Notes in Computer Science, Pittsburgh, Pennsylvania, USA, 2009. Springer.

[7] Armin Scholl. Data of assembly line balancing problems. *Technische Universitt Darmstadt*, 93.

[8] Paul Shaw. A constraint for bin packing. *Principles and Practice of Constraint Programming CP 2004*, 3258/2004:648–662, 2004.