5.53 bin_packing

DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from cumulative.

Constraint bin_packing(CAPACITY, ITEMS)

Arguments

CAPACITY : int
ITEMS : collection(bin−dvar, weight−int)

Restrictions

CAPACITY ≥ 0
required(ITEMS, [bin, weight])
ITEMS.weight ≥ 0
ITEMS.weight $\leq$ CAPACITY

Purpose Given several items of the collection ITEMS (each of them having a specific weight), and different bins of a fixed capacity, assign each item to a bin so that the total weight of the items in each bin does not exceed CAPACITY.

Example

$$\left(5, \left\langle \begin{array}{ll} \texttt{bin}-3 & \texttt{weight}-4, \\ \texttt{bin}-1 & \texttt{weight}-3, \\ \texttt{bin}-3 & \texttt{weight}-1 \end{array} \right\rangle\right)$$

The bin_packing constraint holds since the sum of the height of items that are assigned to bins 1 and 3 is respectively equal to 3 and 5. The previous quantities are both less than or equal to the maximum CAPACITY 5. Figure 5.120 shows the solution associated with the example.

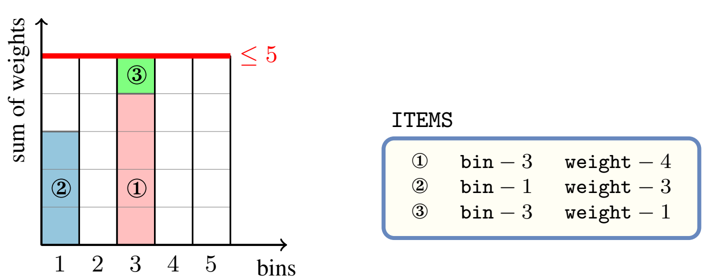


Figure 5.120: Bin-packing solution to the **Example** slot

Typical

CAPACITY $>$ maxval(ITEMS.weight)
CAPACITY $\leq$ sum(ITEMS.weight)
$|$ITEMS$| > 1$
range(ITEMS.bin) > 1
range(ITEMS.weight) > 1
ITEMS.bin ≥ 0
ITEMS.weight > 0

Symmetries

- CAPACITY can be increased.
- Items of ITEMS are permutable.
- ITEMS.weight can be decreased to any value ≥ 0.
- All occurrences of two distinct values of ITEMS.bin can be swapped; all occurrences of a value of ITEMS.bin can be renamed to any unused value.

Arg. properties

Contractible wrt. ITEMS.

Remark

Note the difference with the *classical* bin-packing problem [275, page 221] where one wants to find solutions that minimise the number of bins. In our case each item may be assigned only to specific bins (i.e., the different values of the bin variable) and the goal is to find a feasible solution. This constraint can be seen as a special case of the cumulative constraint [1], where all task durations are equal to 1.

In [379] the CAPACITY parameter of the bin_packing constraint is replaced by a collection of domain variables representing the *load* of each bin (i.e., the sum of the weights of the items assigned to a bin). This allows representing problems where a minimum level has to be reached in each bin.

Coffman *and al.* give in [119] the worst case bounds of different list algorithms for the bin packing problem (i.e., given a positive integer CAPACITY and a list L of integer sizes $\mathtt{weight}_1, \mathtt{weight}_2, \ldots, \mathtt{weight}_n$ $(0 \leq \mathtt{weight}_i \leq \mathtt{CAPACITY})$, what is the smallest integer m such that there is a partition $L = L_1 \cup L_2 \cup \cdots \cup L_m$ satisfying $\sum_{\mathtt{weight}_i \in L_j} \mathtt{weight}_i \leq \mathtt{CAPACITY}$ for all $j \in [1, m]$?).

Algorithm

Initial filtering algorithms are described in [291, 288, 289, 290, 379]. More recently, linear continuous relaxations based on the graph associated with the dynamic programming approach for knapsack by Trick [408], and on the more compact model introduced by Carvalho [101, 102] are presented in [89].

Systems

pack in **Choco**, binpacking in **Gecode**, bin_packing in **MiniZinc**.

See also

generalisation: bin_packing_capa *(fixed overall capacity replaced by non-fixed capacity)*, cumulative *(task of* duration 1 *replaced by task of given* duration*)*, cumulative_two_d *(*task *of* duration 1 *replaced by* square *of size* 1 *with a* height*)*, indexed_sum *(negative contribution also allowed, fixed capacity replaced by a set of variables)*.

used in graph description: sum_ctr.

Keywords

application area: assignment.

characteristic of a constraint: automaton, automaton with array of counters.

constraint type: resource constraint.

final graph structure: acyclic, bipartite, no loop.

modelling: assignment dimension, assignment to the same set of values.

modelling exercises: assignment to the same set of values.

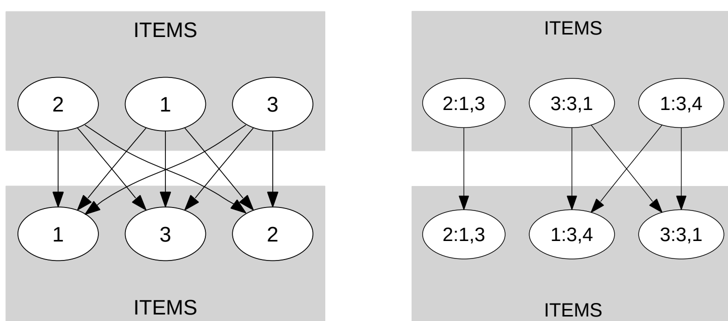
Cond. implications

bin_packing(CAPACITY, ITEMS)
with CAPACITY $\geq$ |ITEMS|
implies atmost_nvector(NVEC : CAPACITY, VECTORS : ITEMS).

Arc input(s)	`ITEMS ITEMS`
Arc generator	$PRODUCT \mapsto$ `collection(items1, items2)`
Arc arity	2
Arc constraint(s)	`items1.bin` $=$ `items2.bin`
Graph class	• `ACYCLIC` • `BIPARTITE` • `NO_LOOP`
Sets	$\text{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var} - \texttt{ITEMS.weight})]\end{array}\right)\end{array}\right]$
Constraint(s) on sets	`sum_ctr(variables,` $\leq$`, CAPACITY)`

Graph model We enforce the `sum_ctr` constraint on the weight of the items that are assigned to the same bin.

Parts (A) and (B) of Figure 5.121 respectively show the initial and final graph associated with the **Example** slot. Each connected component of the final graph corresponds to the items that are all assigned to the same bin.



(A) **(B)**

Figure 5.121: Initial and final graph of the `bin_packing` constraint

Automaton Figure 5.122 depicts the automaton associated with the `bin_packing` constraint. To each item of the collection `ITEMS` corresponds a signature variable S_i that is equal to 1.

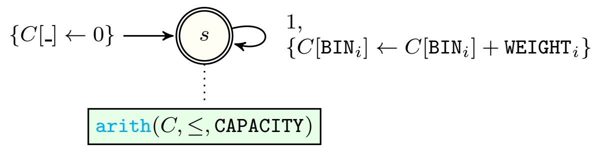


Figure 5.122: Automaton of the `bin_packing` constraint