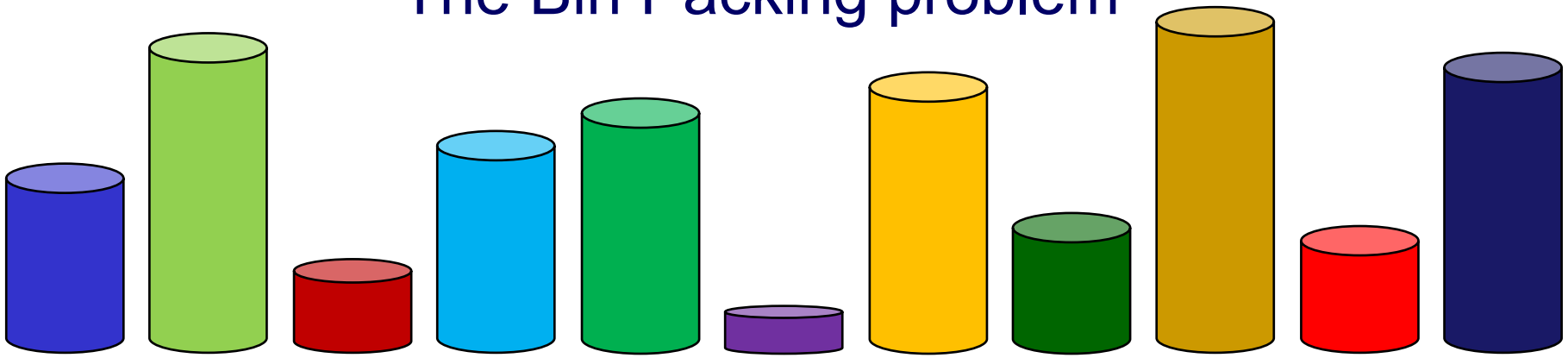
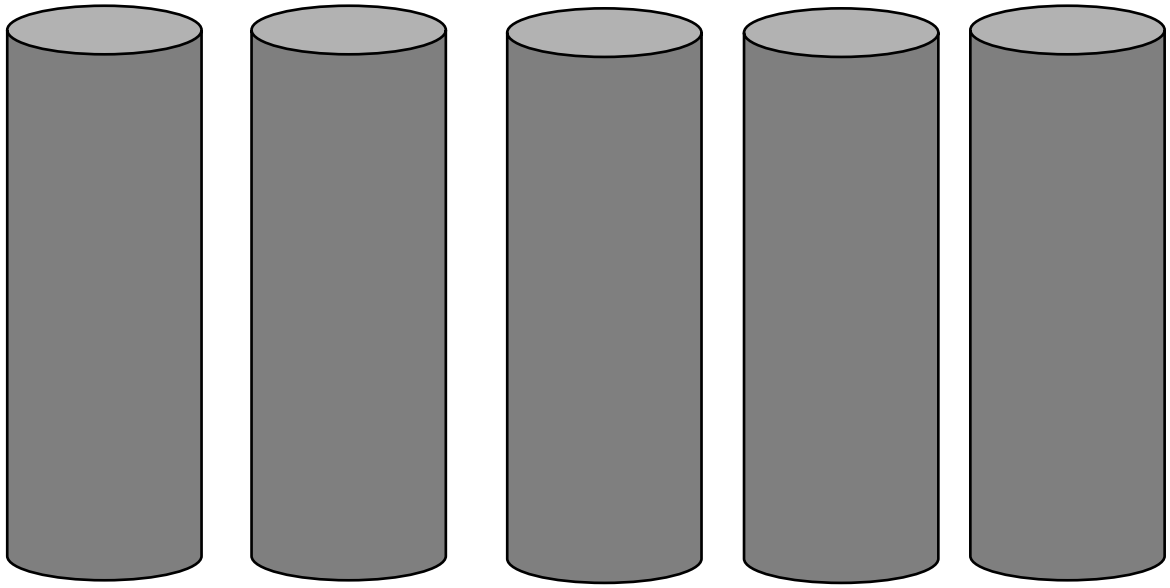
Modelling IV: bin packing

The Bin Packing problem

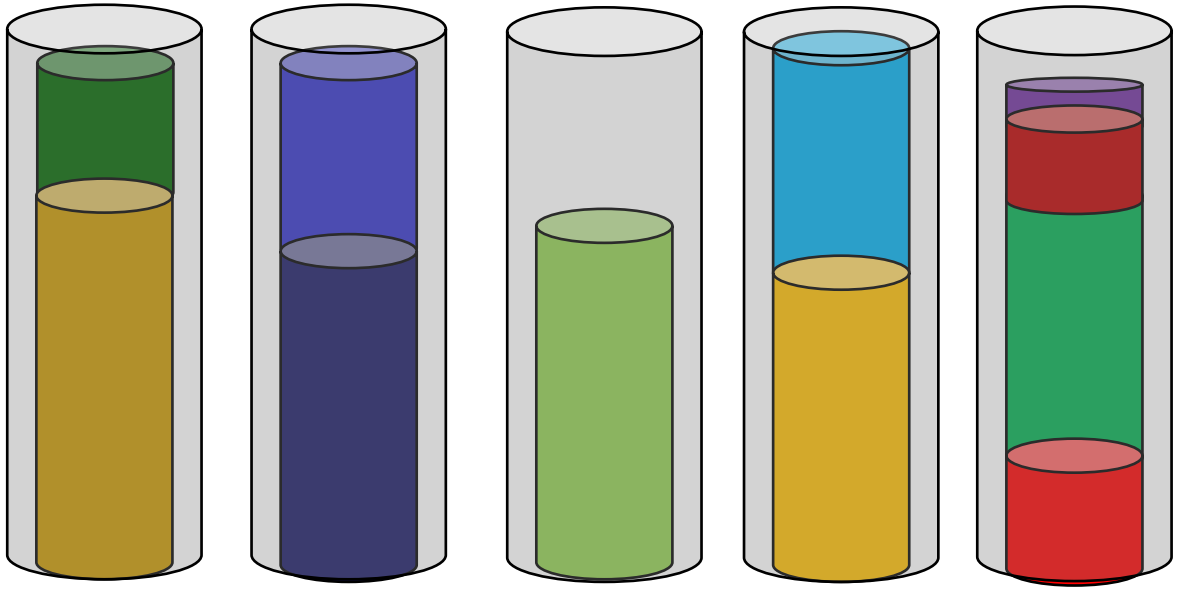




Can we fit these objects into these 5 identical bins?

The Bin Packing problem

(yes)



CPU load balancing

Given a set of *n* processes, each with different running times, and a set of *k* processors, can we schedule all the processes so that all are complete within *t* seconds?

Disk Storage

Given a set of *n* movie files, each with different file sizes, and a set of *k* hard disks, all of the same capacity, can we store all the files on the disks?

How many disks do we need?

The bin packing decision problem

Input: $[s_0, s_1, s_2, ..., s_{n-1}]$ *n* object sizes
k the number of bins
c the (uniform) bin capacity

Problem:
Find an assignment of objects to bins, so that

- no bin contains objects of a total size > c,
- every object is assigned to a bin

First Representation

bin_i contains $object_j$ if and only if table[i][j] == 1

	obj 0	obj 1	obj 2	...	obj (n-1)
bin 0	{0,1}	{0,1}	{0,1}		{0,1}
bin 1	{0,1}	{0,1}	{0,1}		{0,1}
...	...				
bin k	{0,1}	{0,1}	{0,1}		{0,1}

each object j appears in exactly one bin

$\Sigma=1$ $\Sigma=1$ $\Sigma=1$ $\Sigma=1$

size 0	size 1	size 2	...	size (n-1)

for each bin i, the sum of the sizes of each object in the bin ≤c

$$\left(\sum_{j=0}^{n-1} table[i][j] * size[j]\right) \leq c$$

Minimise the number of bins used

used[i] == 1
if bin is used

bin_i contains $object_j$ if and only if table[i][j] == 1

	obj 0	obj 1	obj 2	...	obj (n-1)
bin 0	{0,1}	{0,1}	{0,1}		{0,1}
bin 1	{0,1}	{0,1}	{0,1}		{0,1}
...	...				
bin k	{0,1}	{0,1}	{0,1}		{0,1}

used
{0,1}
{0,1}
...
{0,1}

object j appears in bin i only if bin i is used

$\Sigma=1$ $\Sigma=1$ $\Sigma=1$ $\Sigma=1$ $\Sigma=X$

$table[i][j] \leq used[i]$

size 0	size 1	size 2	...	size (n-1)

minimisation will stops bins being 'used' but having no objects

$$\left(\sum_{j=0}^{n-1} table[i][j] * size[j]\right) \leq c$$

minimise X

Optimisation: alternative model

Minimise the number of bins required

$$\text{load}_i = \left(\sum_{j=0}^{n-1} table[i][j] * size[j]\right)$$

$$\text{used}_i == 1 <==> \text{load}_i > 0$$

Logical expression over two constraints

$$\text{minimise} \sum_{i=0}^{k-1} used_i$$

Reified Constraints

A constraint is *reified* if a boolean variable is created which states whether or not the constraint is satisfied.

The constraint itself is not automatically posted in the solver.

E.g. *b* is a Boolean variable (domain = {0,1}), and
c is an integer variable with domain {0,1,...,9}

$$b = 1 \iff c > 3$$

If *c* takes a value > 3, then *b* will have value 1
If *c* takes a value ≤ 3, then *b* will have value 0
If *b* takes value 0, then *c*'s domain is reduced to {0,1,2,3}
If *b* takes value 1, then *c*'s domain is reduced to {4,5,...,9}

LogicalConstraintFactory

"The LogicalConstraintFactory (or LCF) provides various interesting constraints to manipulate other constraints. These constraints are based on the concept of reification. We say a constraint C is reified with a boolean variable b when we maintain the equivalence between b being equal to true and C being satisfied. This means the C constraint may be not satisfied, hence it should not be posted to the solver."

```
IntVar v1 = VariableFactory.enumerated("v1", 0, 3, solver);
Constraint c1 = IntConstraintFactory.arithm(v1, "<", 2);

IntVar v2 = VariableFactory.enumerated("v2", 0, 3, solver);
Constraint c2 = IntConstraintFactory.arithm(v2, ">", 1);

LogicalConstraintFactory.ifThen(c1, c2);

BoolVar b1 = VariableFactory.bool("b1", solver);
LogicalConstraintFactory.reification(b1, c1);

Chatterbox.showSolutions(solver);
solver.findAllSolutions();
```

What solutions do you expect?

The bin packing constraint

18.9 bin_packing

The *bin_packing* constraint involves:

- an array of integer variables *ITEM_BIN*,
- an array of integers *ITEM_SIZE*,
- an array of integer variables *BIN_LOAD* and
- an integer *OFFSET*.

It holds the Bin Packing Problem rules: a set of items with various SIZES to pack into bins with respect to the capacity of each bin.

- *ITEM_BIN* represents the bin of each item, that is, *ITEM_BIN[i] = j* states that the i th ITEM is put in the j th bin.
- *ITEM_SIZE* represents the size of each item.
- *BIN_LOAD* represents the load of each bin, that is, the sum of size of the items in it.

This constraint is not a built-in constraint and is based on various propagators.

See also: bin_packing in the Global Constraint Catalog.

Second representation

binForObject	obj 0	obj 1	obj 2	...	obj (n-1)
bin?	{0,...,k-1}	{0,...,k-1}	{0,...,k-1}		{0,...,k-1}

Choco offers a global constraint:

bin_packing(binForObject, sizes, binLoad)

Exercise

Experiment with the Choco classes, varying the problem data.

Does changing the search heuristics change the solving time?

If you were solving this by hand, what heuristics (i.e. what variable or value to try next) might be good??

Next lecture ...

more modeling: scheduling