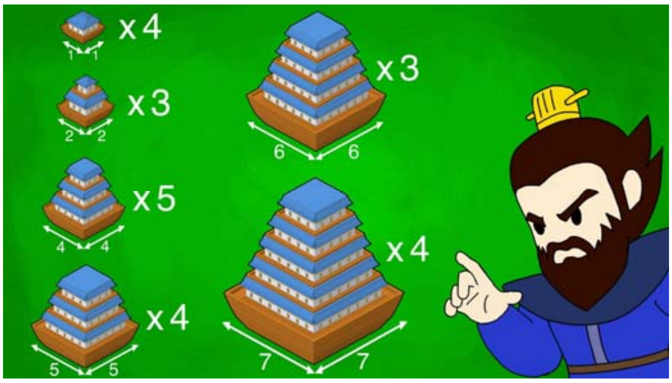
THE UNIVERSITY OF MELBOURNE 香港中文大學 The Chinese University of Hong Kong

Square Packing

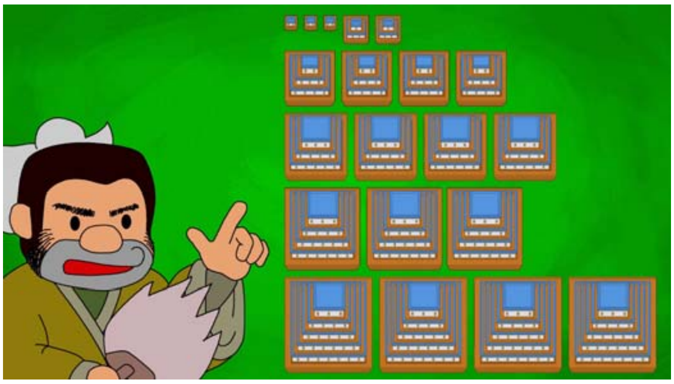
Jimmy Lee & Peter Stuckey

Packing House Ships



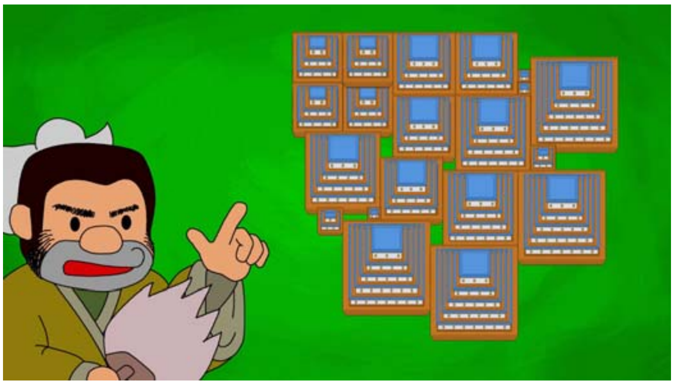
2

House Ships as Squares



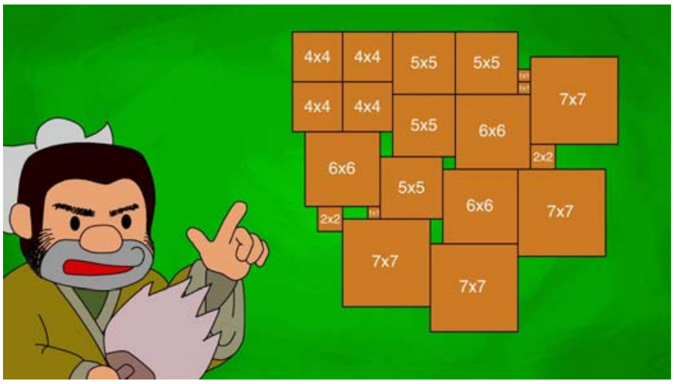
3

House Ships as Squares



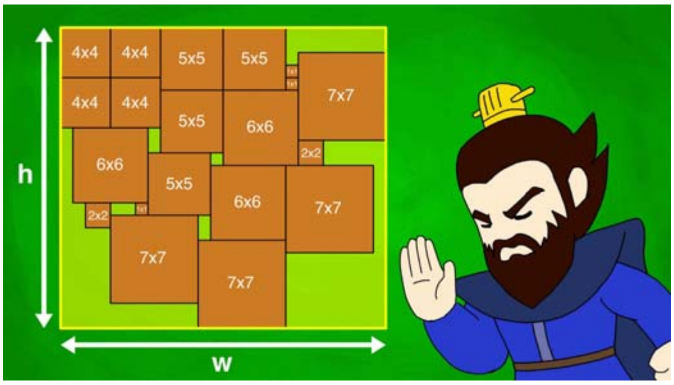
4

House Ships as Squares



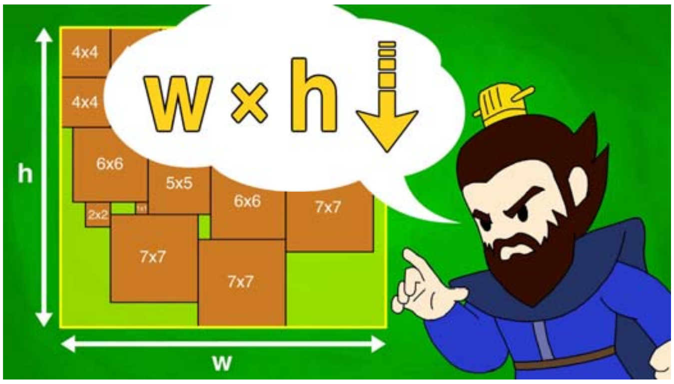
5

House Ships as Squares



6

House Ships as Squares



7

Multiple Square Packing

- Given
  - k_1 1x1 squares
  - k_2 2x2 squares
  - …
  - k_n nxn squares
- Pack the squares into a rectangle of the smallest area

8

Data and Variables (multisqpack.mzn)

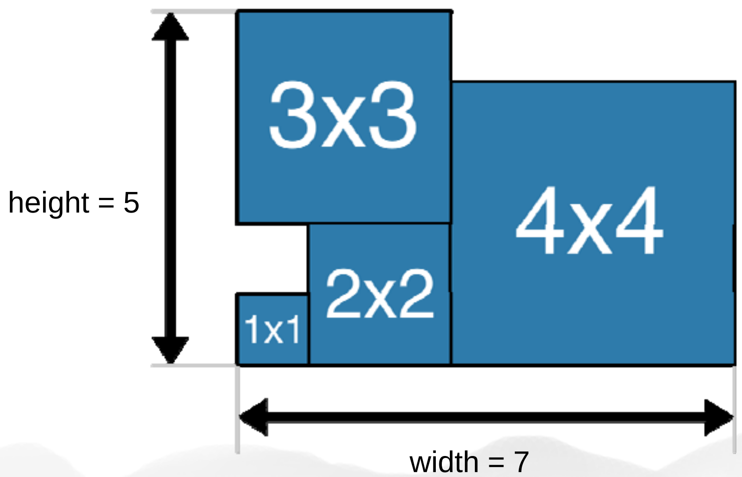
```
int: n; % number of square sizes
set of int: SQUARE = 1..n;
array[SQUARE] of int: ncopy;
int: maxl = sum(i in SQUARE)(i*ncopy[i]);
int: mina = sum(i in SQUARE)(i*i*ncopy[i]);

var n..maxl: height; var n..maxl: width;
var mina .. n*maxl: area = height * width;
int: nsq = sum(i in SQUARE)(ncopy[i]);
set of int: NSQ = 1..nsq;
array[NSQ] of var 0..maxl: x;
array[NSQ] of var 0..maxl: y;
```

- Note the tight bounds on variables

9

(Single) Square Packing Solution



area = 5x7 = 35

10

Auxiliary Variables (multisqpack.mzn)

- Useful auxiliary variables

```
array[NSQ] of var SQUARE: size;
% calculate size of each square
include "global_cardinality.mzn";
global_cardinality(size,
   [i | i in SQUARE], ncopy);
forall(i in 1..nsq-1)
   (size[i] <= size[i+1]);
```

- For example

```
ncopy = [3,2,5,4,3]
size = [1,1,1,2,2,3,3,3,3,3,4,4,4,4,5,5,5]
```

11

Constraints & Objective (multisqpack.mzn)

- Squares fit in the rectangle

```
forall(s in NSQ)(x[s] + size[s] <= width);
forall(s in NSQ)(y[s] + size[s] <= height);
```

- Squares do not overlap

```
forall(s1, s2 in NSQ where s1 < s2)
   (x[s1] + size[s1] <= x[s2] \/
    x[s2] + size[s2] <= x[s1] \/
    y[s1] + size[s1] <= y[s2] \/
    y[s2] + size[s2] <= y[s1]);
```

- Objective

```
solve minimize area;
```

12

Solving the Model

- A toy instance

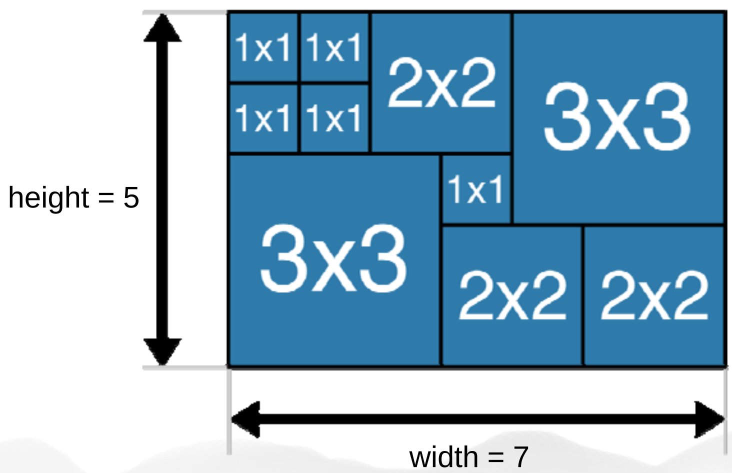
```
n = 3;
ncopy = [5,3,2];
```

- Output

```
area = 35
height = 5
width = 7
x = [1, 1, 0, 0, 3, 3, 2, 5, 4, 0]
y = [4, 3, 3, 4, 2, 0, 3, 0, 2, 0]
----------
==========
Finished in 49msec
```

13

A Solution for the Toy Problem



area = 5x7 = 35

14

Solving the Model Again

- The real instance

```
n = 7;
ncopy = [4,3,0,5,4,3,4];
```

- After 6s

```
area = 520
…
```

- After 1m

```
area = 507
…
```

- After 7m and up to 1h

```
area = 504
…
```

15

Improving the Model

- Global constraints
- Redundant constraints
- Symmetry breaking

16

The diffn Global Constraint

- The diffn global constraint captures exactly 2d non overlap (it should be called diff2)
  - diffn([x_1, …, x_n], [y_1, …, y_n],
  - [dx_1, …, dx_n], [dy_1, …, dy_n])
    - ensure no two objects at positions (x_i,y_i) with dimensions (dx_i,dy_i) overlap

```
predicate diffn(array[int] of var int: x,
                array[int] of var int: y,
                array[int] of var int: dx,
                array[int] of var int: dy);
```

- Squares do not overlap `(multisqpackimp.mzn)`

```
include "diffn.mzn";
diffn(x, y, size, size);
```

17

Packing and Cumulative

- If there is a packing
  - then the cumulative constraint must hold!
- We can add **redundant** cumulative constraints to packing problems
  - improves propagation (and hence solving)

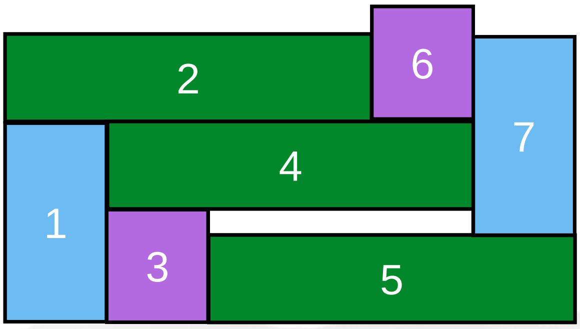
- Squares do not overlap in the x and y dimension respectively `(multisqpackimp.mzn)`

```
cumulative(x, size, size, height);
cumulative(y, size, size, width);
```

18

Packing and Cumulative

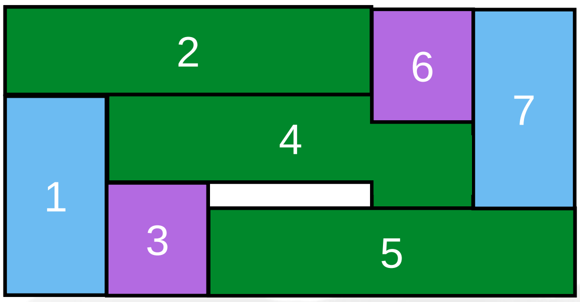
- In general
  - cumulative constraints do not enforce packing
  - even when the the x positions are fixed



19

Packing and Cumulative

- In general
  - cumulative constraints do not enforce packing
  - even when the the x positions are fixed



20

Symmetries!

- Squares of the same size are interchangeable, creating multiplicity of solution possibilities
- Impose an ordering on the placements of such squares
- What ordering can we use for coordinates (x,y)?
- Strict lexicographical ordering
  - $(x_1,y_1) >_{lex} (x_2,y_2)$
    - $x_1 > x_2$; or
    - if $x_1 = x_2$, then $y_1 > y_2$

21

The lex_greater Global Constraint

- The lex_greater global constraint imposes the lexicographic ordering on two *n*-tuples (encoded as arrays)
  - lex_greater([x_1, …, x_n], [y_1, …, y_n])
    - ensures that $(x_1,\ldots,x_n) >_{lex} (y_1,\ldots,y_n)$

```
predicate lex_greater(array [int] of var int: x,
                      array [int] of var int: y)
```

22

Ordering Squares

- The placement of a square is specified by the coordinates of its lower left hand corner
- Find the starting index of each size

```
array[SQUARE] of int: base =
   [if i = 1 then 0 else
      sum(j in 1..i-1)(ncopy[j]) endif
    | i in SQUARE];
```

- Order squares of the same size

```
include "lex_greater.mzn";
constraint forall(i in SQUARE)
   (forall(j in 1..ncopy[i]-1)(
      lex_greater([x[base[i]+j],y[base[i]+j]],
         [x[base[i]+j+1],y[base[i]+j+1]])));
```

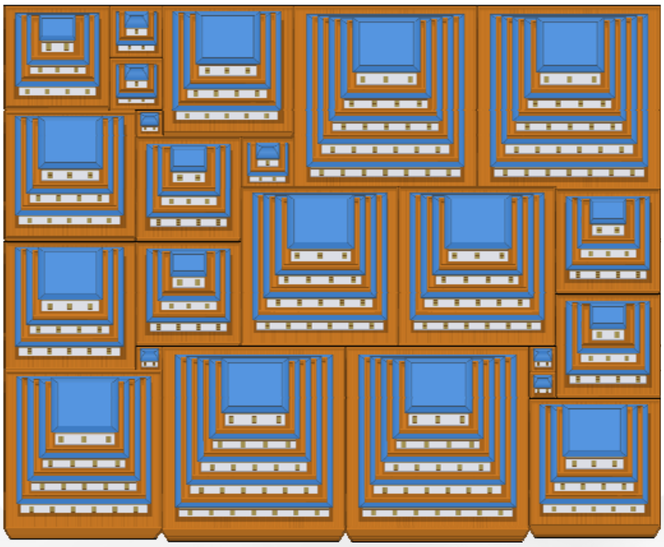
23

Solving the Model Again

```
area = 500
height = 20
width = 25
x = [20, 20, 5, 5, 9, 4, 4, 21, 21, 5, 5,
0, 20, 6, 0, 0, 15, 9, 0, 18, 13, 11, 6]
y = [6, 5, 15, 6, 13, 18, 16, 9, 5, 11, 7,
16, 0, 15, 11, 6, 7, 7, 0, 13, 0, 13, 0]
----------
==========
Finished in 32s 846msec
```

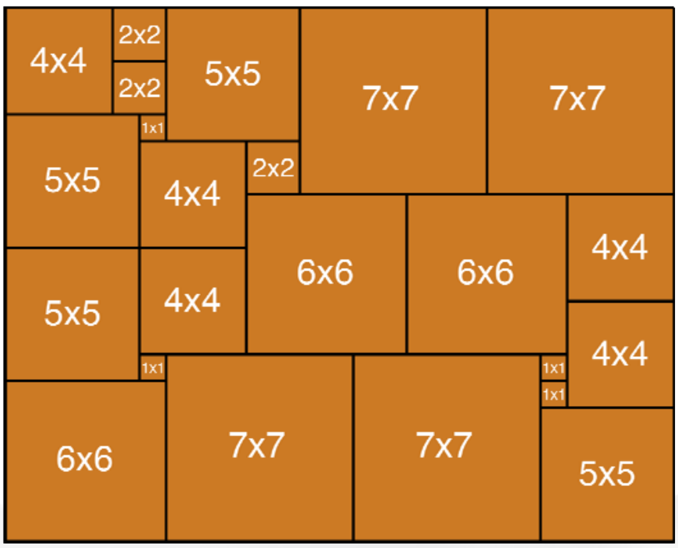
24

Solving the Model Again



25

Solving the Model Again



26

Another Improvement (multisqpack.mzn)

- The array size is a variable solved by constraints

```
array[NSQ] of var SQUARE: size;
include "global_cardinality.mzn";
global_cardinality(size,
   [i | i in SQUARE], ncopy);
forall(i in 1..nsq-1)
   (size[i] <= size[i+1]);
```

- Can do without constraint solving

```
array[NSQ] of SQUARE: size;
size = [max(j in SQUARE)(j*(i > base[j]))
   | i in NSQ];
```

27

Summary

- Packing problems
  - are another common uses of CP in the real world
  - come in lots of varieties
  - are complex discrete optimization problems
- diffn encodes 2D non-overlap
  - disjunctive encodes 1D non-overlap
- cumulative constraints are redundant for packing
  - but useful for improving solving

28

Image Credits

All graphics by Marti Wong, ©The Chinese University of Hong Kong and the University of Melbourne 2016

29