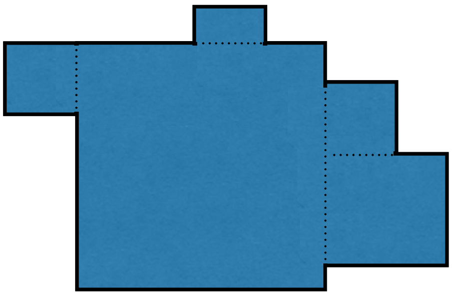
Rectilinear Packing with Rotation

Jimmy Lee & Peter Stuckey

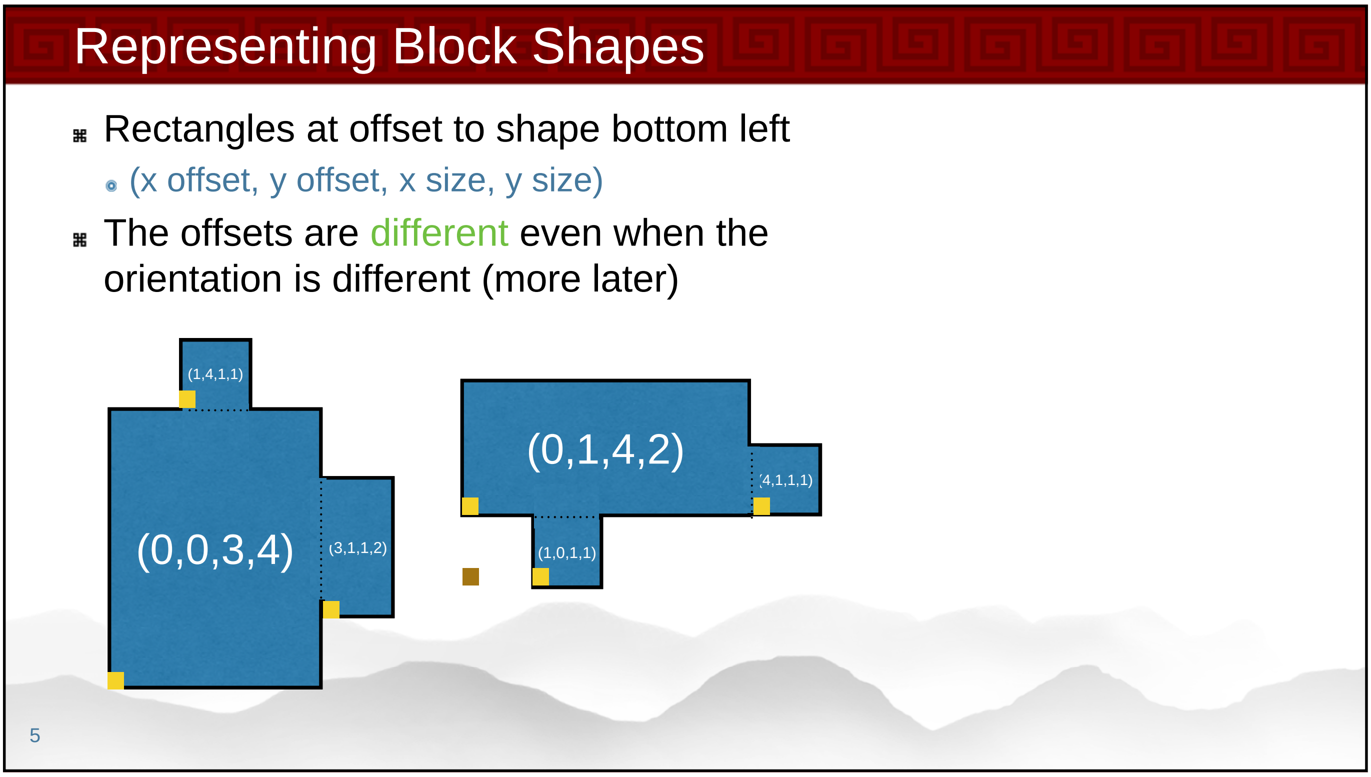
Rotations

A Shape Usually Has 4 Orientations



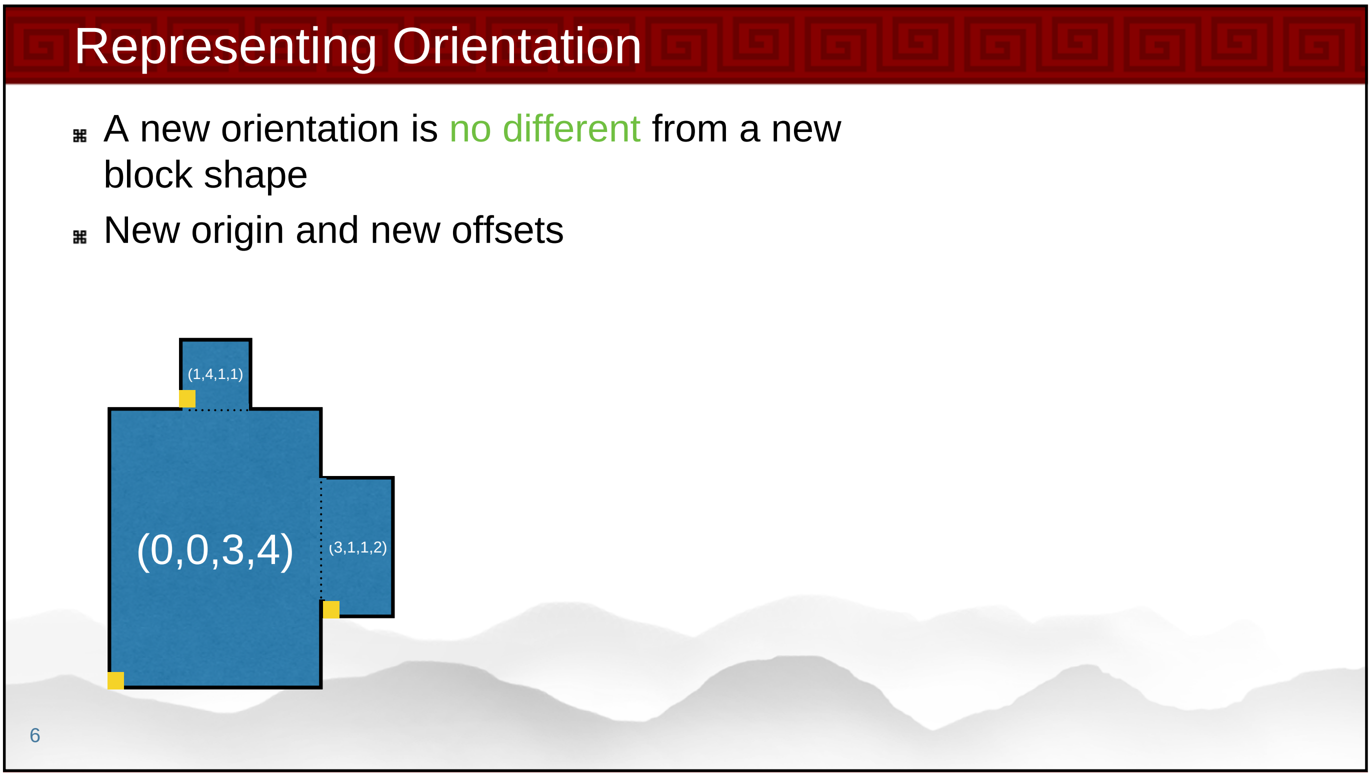
Orientation Exceptions

- The shape is a plain rectangle
  - 2 orientations only
- The shape is a square
  - 1 orientation only
- Or application-oriented restrictions
  - A block of ships can move only in specific directions



Representing Block Shapes

- Rectangles at offset to shape bottom left
  - (x offset, y offset, x size, y size)
- The offsets are different even when the orientation is different (more later)

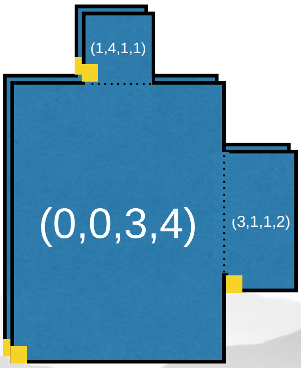


Representing Orientation

- A new orientation is no different from a new block shape
- New origin and new offsets

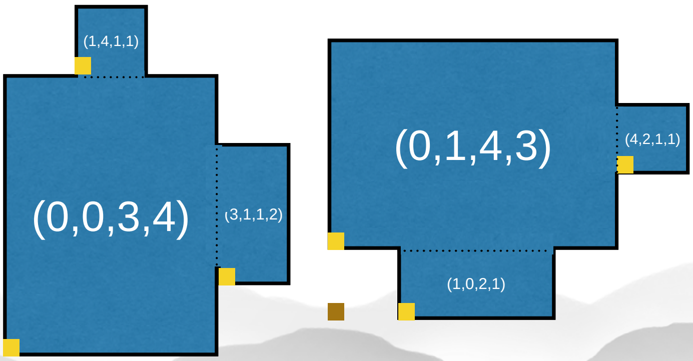
Representing Orientation

- A new orientation is no different from a new block shape
- New origin and new offsets



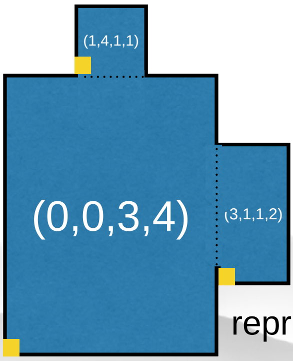
Representing Orientation/Rotation

- A new orientation is no different from a new block shape
- New origin and new offsets



Representing Block Shapes

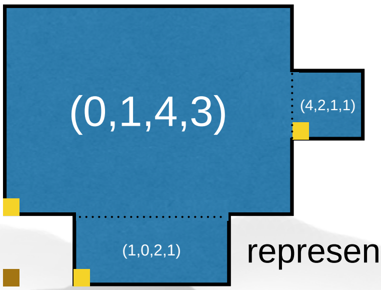
- Number the rectangle offsets
- A block (of a specific orientation) is
  - a set of rectangles with offsets
- Rectangle offsets can be shared if possible
- E.g. a list of offsets (ordering unimportant)
  - **1: 0,0,3,4**
  - 2: 0,1,4,3
  - **3: 1,4,1,1**
  - **4: 3,1,1,2**
  - 5: 4,2,1,1
  - 6: 1,0,2,1



representation {1,3,4}

Representing Block Shapes

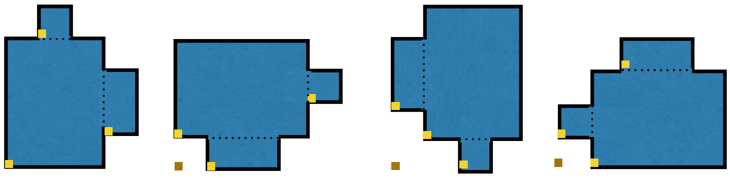
- Number the rectangle offsets
- A block (of a specific orientation) is
  - a set of rectangles with offsets
- Rectangle offsets can be shared if possible
- E.g. a list of offsets (ordering unimportant)
  - 1: 0,0,3,4
  - **2: 0,1,4,3**
  - 3: 1,4,1,1
  - 4: 3,1,1,2
  - **5: 4,2,1,1**
  - **6: 1,0,2,1**



representation {2,5,6}

Representing a Shape

- A orientation is a set of offsets
- A shape consists of 4 (or less) orientations

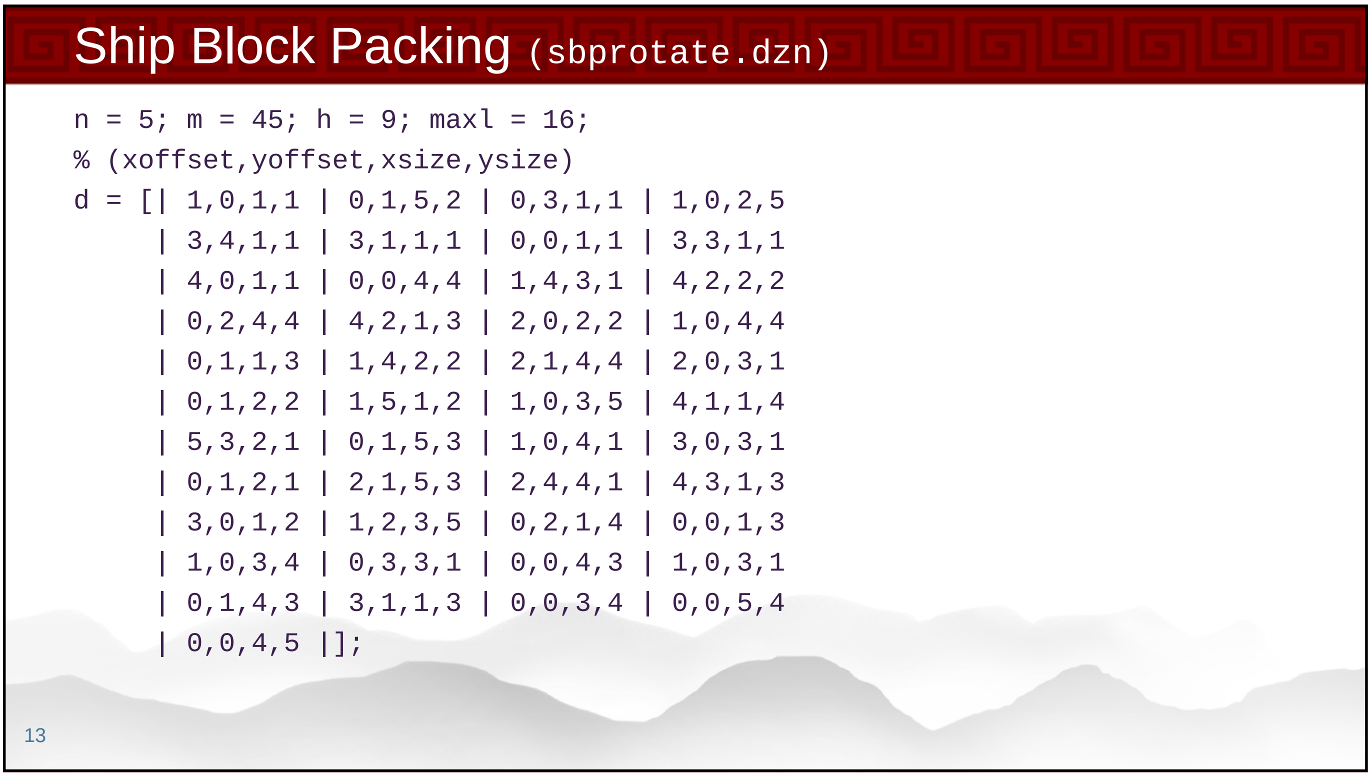


- A shape can thus be represented by a list (array) of sets (of offsets)
- Forbidden/duplicated orientations can be represented by the empty set

Ship Block Packing (sbprotate.mzn)

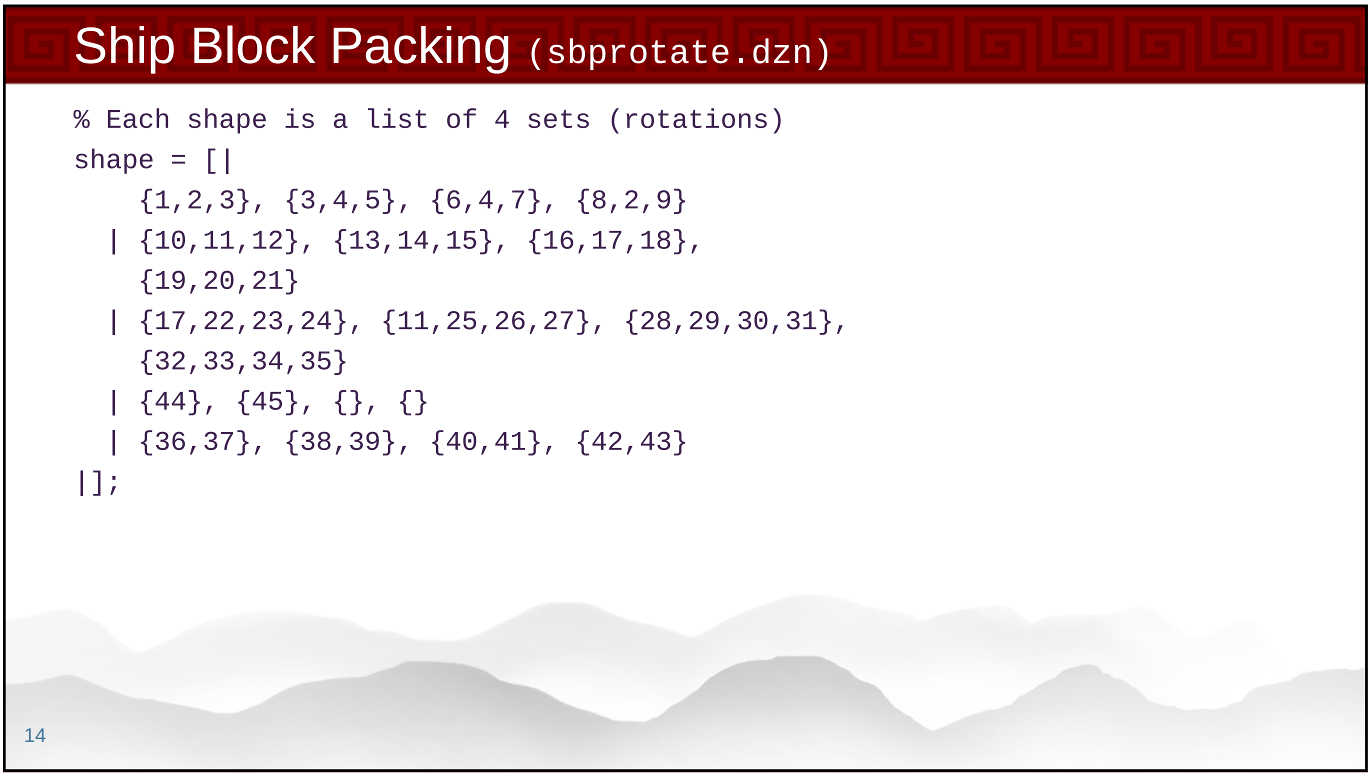
- Given *n* blocks defined by fixed shapes, each with a possible rotations. Place the shapes on a river of width *h* so they don't overlap with the length *l* used minimised

```
int: n; % number of blocks
set of int: BLOCK = 1..n;
int: m; % number of rectangles/offsets
set of int: ROFF = 1..m;
array[ROFF,1..4] of int: d; % defns
set of int: ROT = 1..4;
array[BLOCK,ROT] of set of ROFF: shape;
int: h; % width of river
int: maxl; % maximum length of river
```



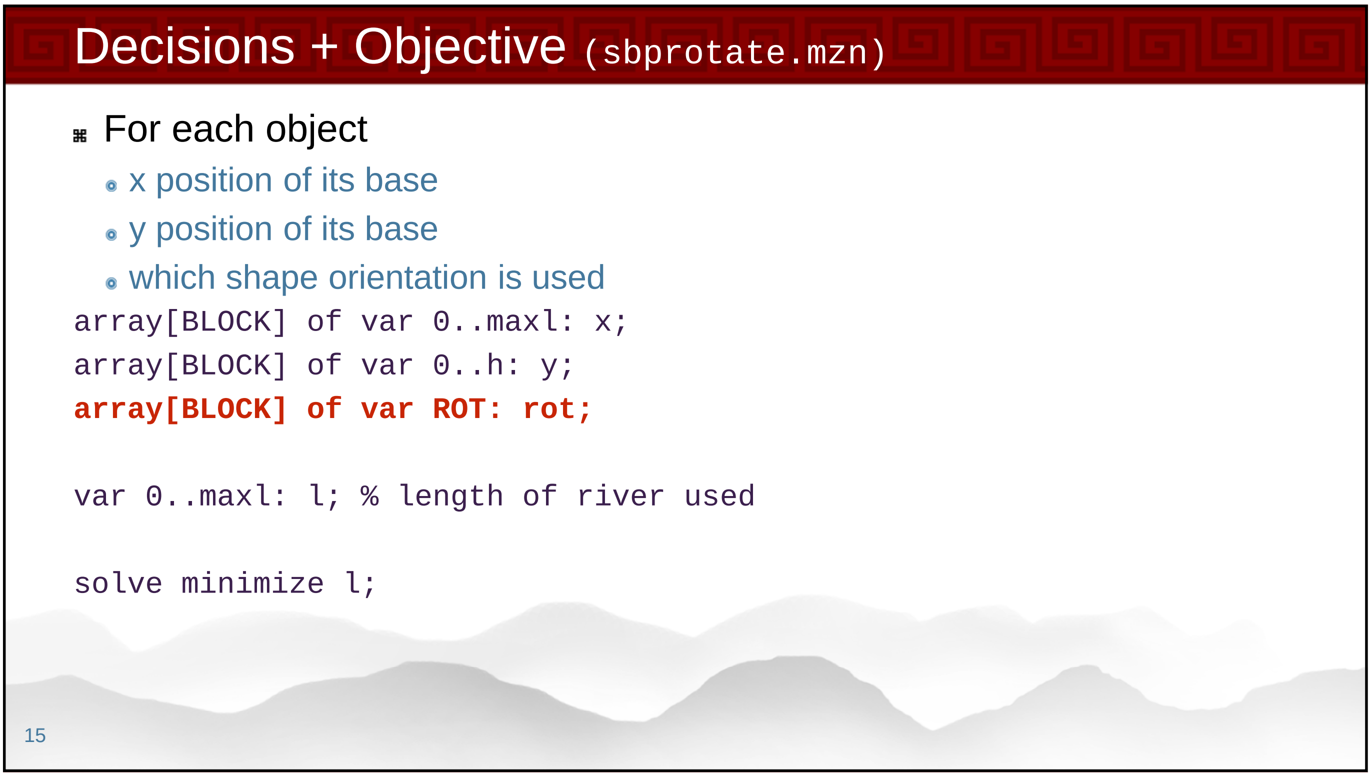
Ship Block Packing (sbprotate.dzn)

```
n = 5; m = 45; h = 9; maxl = 16;
% (xoffset,yoffset,xsize,ysize)
d = [| 1,0,1,1 | 0,1,5,2 | 0,3,1,1 | 1,0,2,5
     | 3,4,1,1 | 3,1,1,1 | 0,0,1,1 | 3,3,1,1
     | 4,0,1,1 | 0,0,4,4 | 1,4,3,1 | 4,2,2,2
     | 0,2,4,4 | 4,2,1,3 | 2,0,2,2 | 1,0,4,4
     | 0,1,1,3 | 1,4,2,2 | 2,1,4,4 | 2,0,3,1
     | 0,1,2,2 | 1,5,1,2 | 1,0,3,5 | 4,1,1,4
     | 5,3,2,1 | 0,1,5,3 | 1,0,4,1 | 3,0,3,1
     | 0,1,2,1 | 2,1,5,3 | 2,4,4,1 | 4,3,1,3
     | 3,0,1,2 | 1,2,3,5 | 0,2,1,4 | 0,0,1,3
     | 1,0,3,4 | 0,3,3,1 | 0,0,4,3 | 1,0,3,1
     | 0,1,4,3 | 3,1,1,3 | 0,0,3,4 | 0,0,5,4
     | 0,0,4,5 |];
```



Ship Block Packing (sbprotate.dzn)

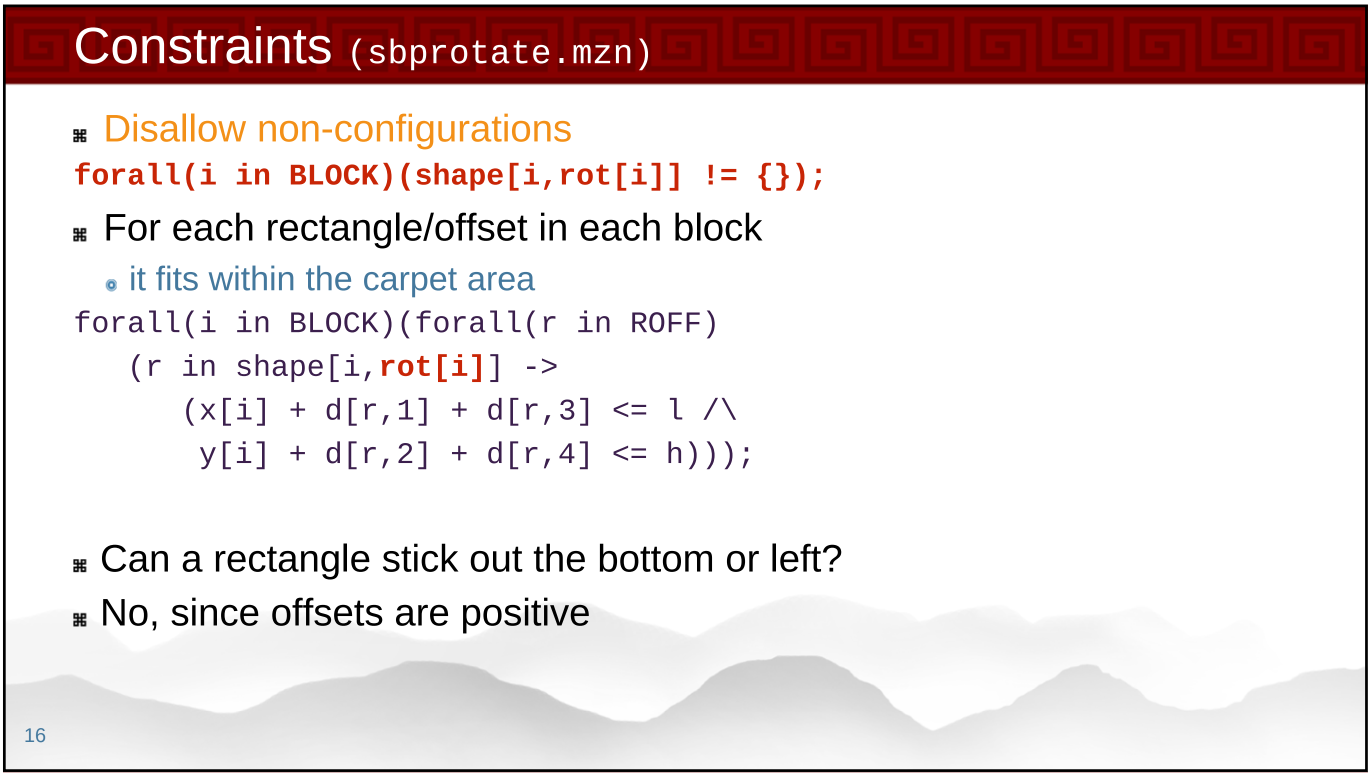
```
% Each shape is a list of 4 sets (rotations)
shape = [|
    {1,2,3}, {3,4,5}, {6,4,7}, {8,2,9}
  | {10,11,12}, {13,14,15}, {16,17,18},
    {19,20,21}
  | {17,22,23,24}, {11,25,26,27}, {28,29,30,31},
    {32,33,34,35}
  | {44}, {45}, {}, {}
  | {36,37}, {38,39}, {40,41}, {42,43}
|];
```



Decisions + Objective (sbprotate.mzn)

- For each object
  - x position of its base
  - y position of its base
  - which shape orientation is used

```
array[BLOCK] of var 0..maxl: x;
array[BLOCK] of var 0..h: y;
array[BLOCK] of var ROT: rot;

var 0..maxl: l; % length of river used

solve minimize l;
```



Constraints (sbprotate.mzn)

- Disallow non-configurations

```
forall(i in BLOCK)(shape[i,rot[i]] != {});
```

- For each rectangle/offset in each block
  - it fits within the carpet area

```
forall(i in BLOCK)(forall(r in ROFF)
   (r in shape[i,rot[i]] ->
      (x[i] + d[r,1] + d[r,3] <= l /\
       y[i] + d[r,2] + d[r,4] <= h)));
```

- Can a rectangle stick out the bottom or left?
- No, since offsets are positive

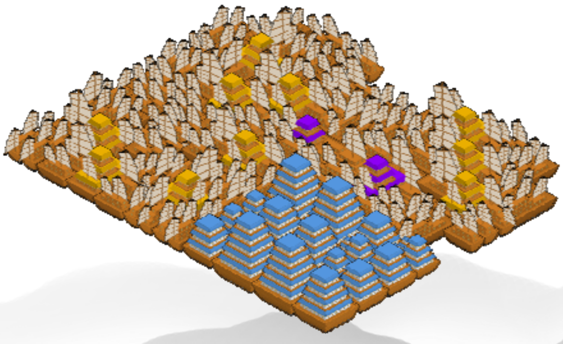
Constraints (sbprotate.mzn)

- Rectangle/offsets don't overlap

```
forall(i,j in BLOCK where i < j)
   (forall(r1,r2 in ROFF)
       (r1 in shape[i,rot[i]] /\
        r2 in shape[j,rot[j]] ->
(x[i] + d[r1,1] + d[r1,3] <= x[j] + d[r2,1]
                   \/
 x[j] + d[r2,1] + d[r2,3] <= x[i] + d[r1,1]
                   \/
 y[i] + d[r1,2] + d[r1,4] <= y[j] + d[r2,2]
                   \/
y[j] + d[r2,2] + d[r2,4] <= y[i] + d[r1,2]
)));
```

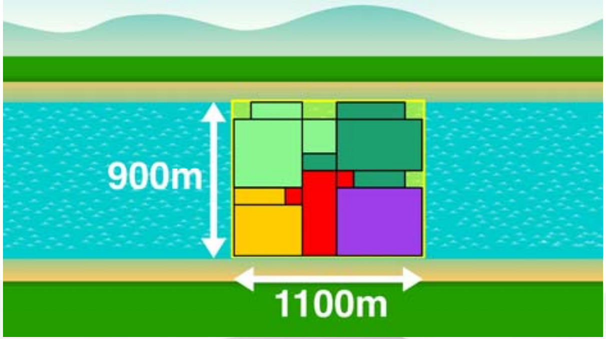
Solving the Model

```
l = 11;
x = [3, 0, 4, 6, 0];
y = [0, 4, 4, 0, 0];
rot = [2, 1, 3, 1, 2];
----------
==========
Finished in 1m 30s
```



Solving the Model

```
l = 11;
x = [3, 0, 4, 6, 0];
y = [0, 4, 4, 0, 0];
rot = [2, 1, 3, 1, 2];
----------
==========
Finished in 1m 30s
```

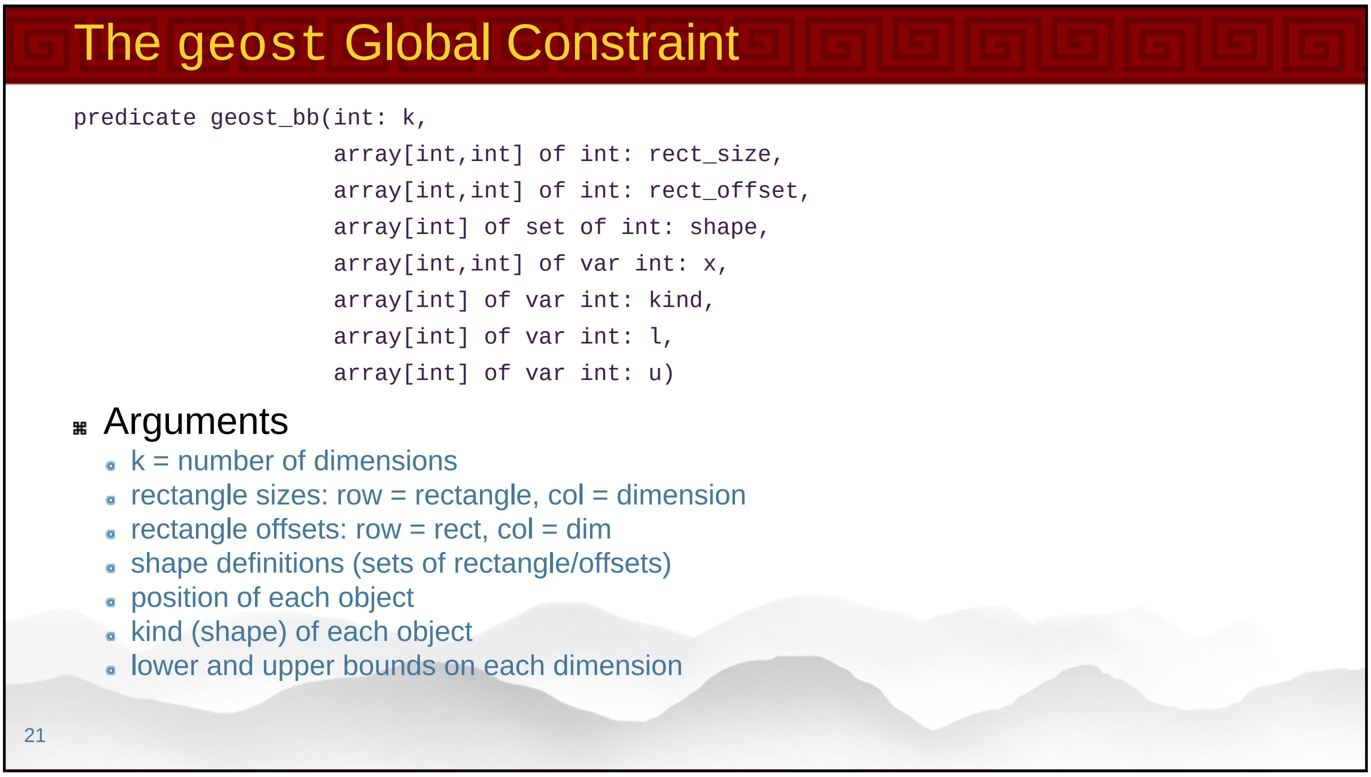


19

Packing Globals

- The global constraint `diffn` is extensible to *k* dimensions
  - in MiniZinc `diffn_k`
- The `geost` global constraint enforces non-overlap of objects, taking rotations into account
  - objects may have multiple possible shapes
  - each shape is a set of offset rectangles

20

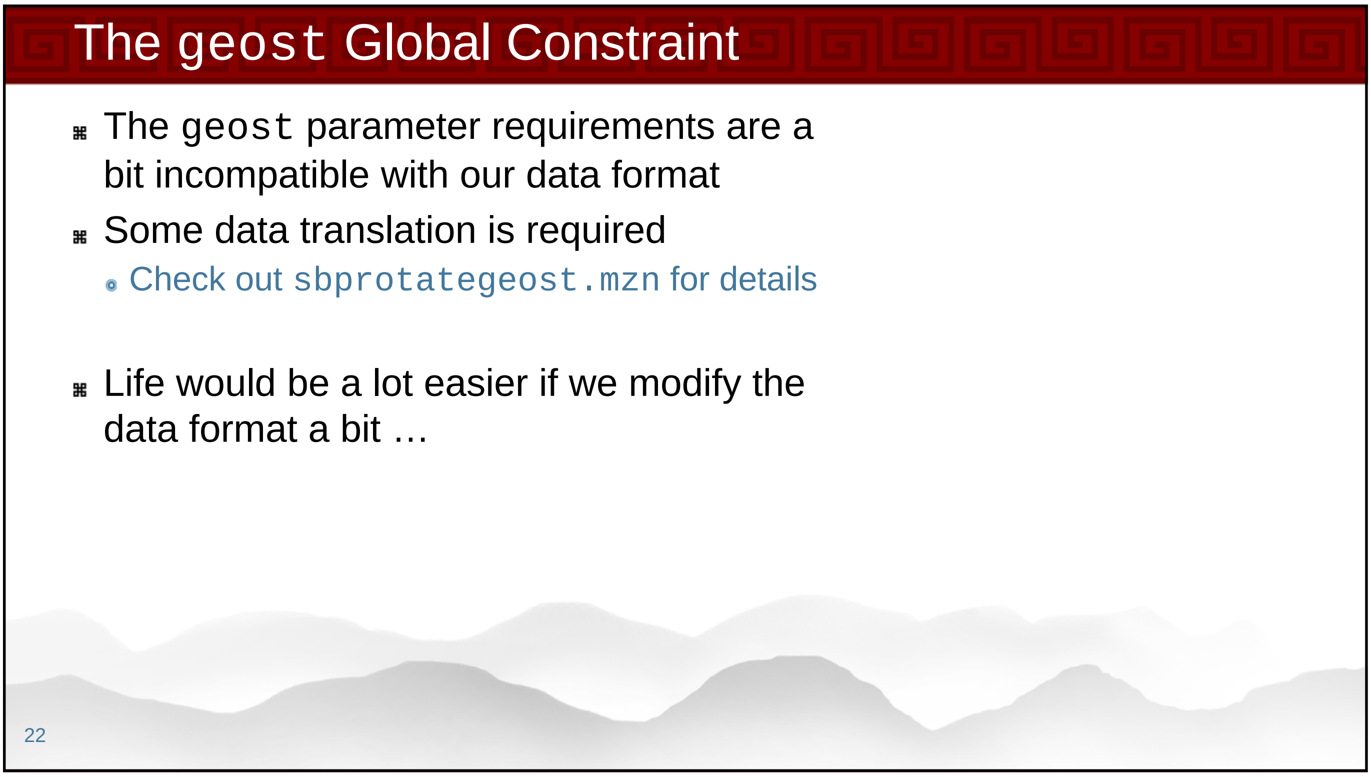


The geost Global Constraint

```
predicate geost_bb(int: k,
                   array[int,int] of int: rect_size,
                   array[int,int] of int: rect_offset,
                   array[int] of set of int: shape,
                   array[int,int] of var int: x,
                   array[int] of var int: kind,
                   array[int] of var int: l,
                   array[int] of var int: u)
```

- Arguments
  - k = number of dimensions
  - rectangle sizes: row = rectangle, col = dimension
  - rectangle offsets: row = rect, col = dim
  - shape definitions (sets of rectangle/offsets)
  - position of each object
  - kind (shape) of each object
  - lower and upper bounds on each dimension

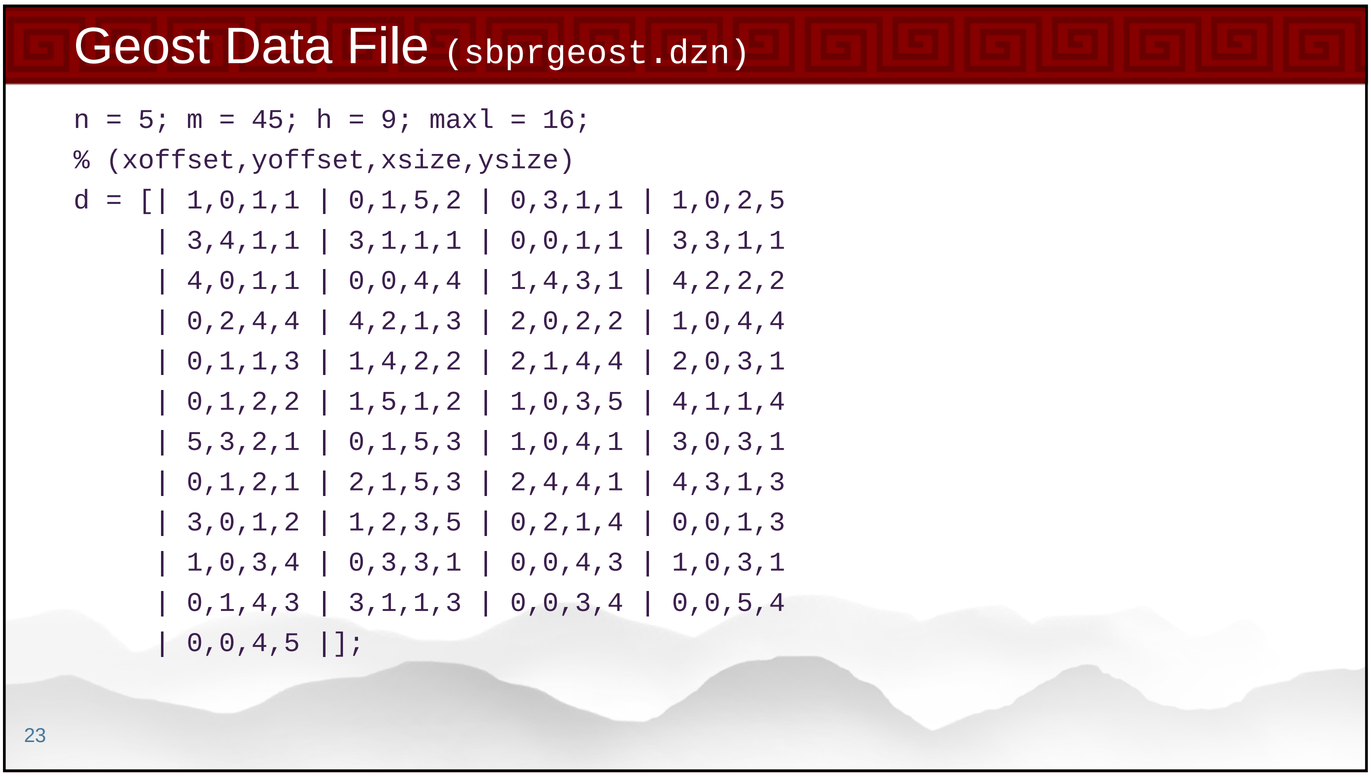
21



The `geost` Global Constraint

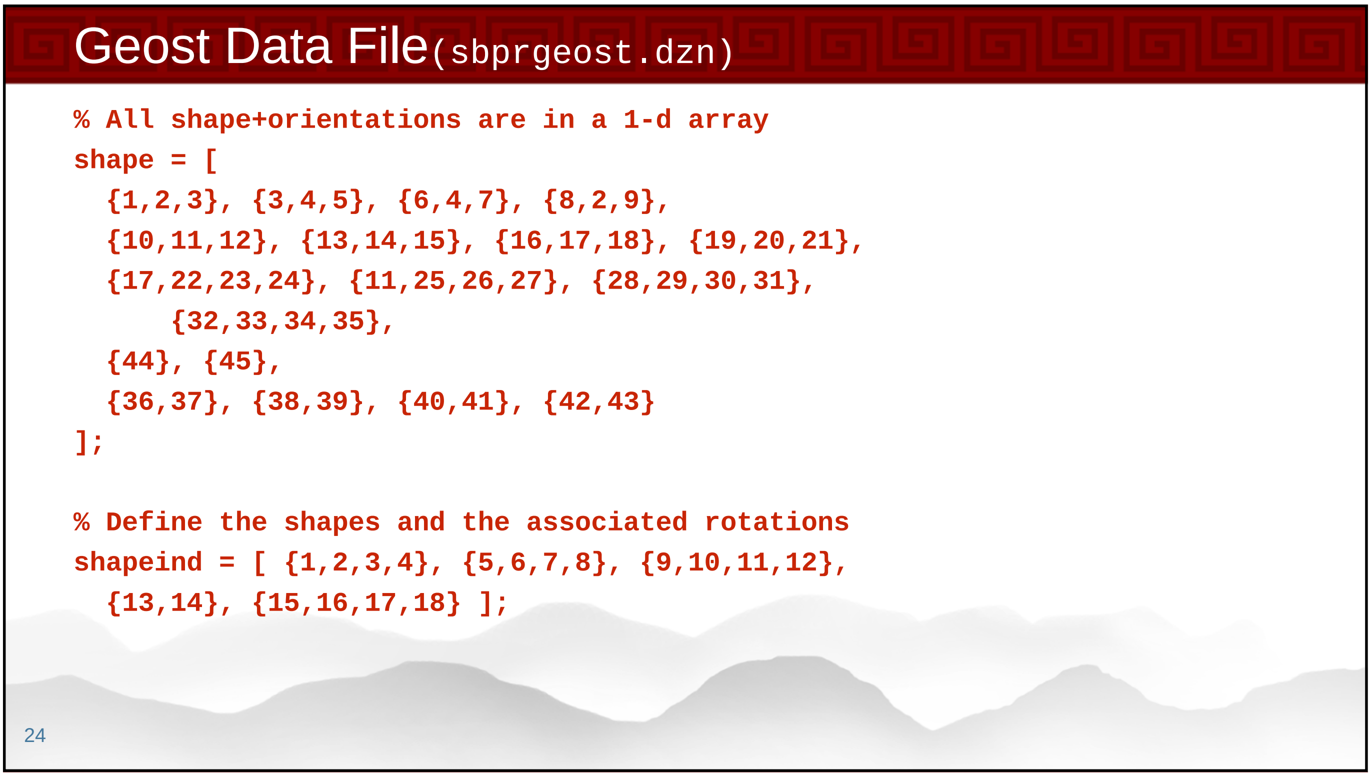
- The `geost` parameter requirements are a bit incompatible with our data format
- Some data translation is required
  - Check out `sbprotategeost.mzn` for details
- Life would be a lot easier if we modify the data format a bit …

22



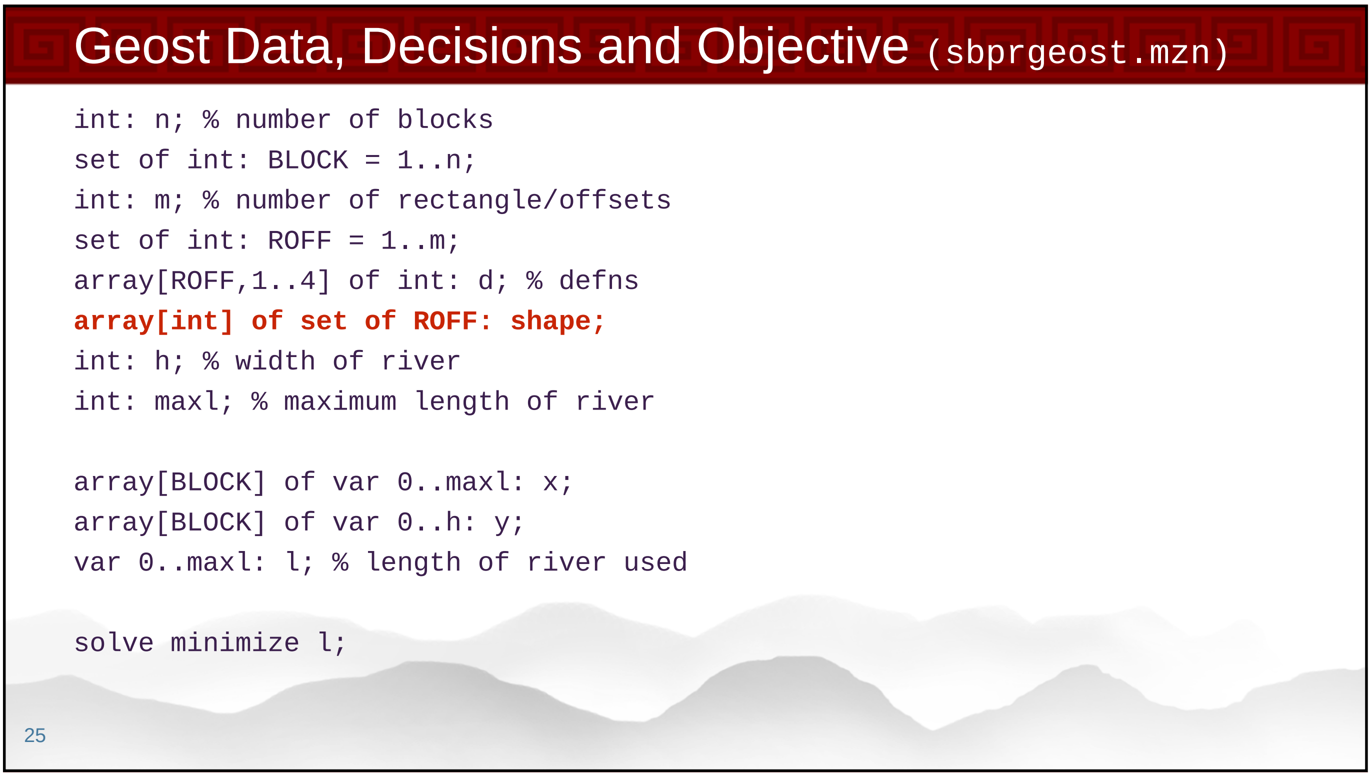
Geost Data File (sbprgeost.dzn)

```
n = 5; m = 45; h = 9; maxl = 16;
% (xoffset,yoffset,xsize,ysize)
d = [| 1,0,1,1 | 0,1,5,2 | 0,3,1,1 | 1,0,2,5
     | 3,4,1,1 | 3,1,1,1 | 0,0,1,1 | 3,3,1,1
     | 4,0,1,1 | 0,0,4,4 | 1,4,3,1 | 4,2,2,2
     | 0,2,4,4 | 4,2,1,3 | 2,0,2,2 | 1,0,4,4
     | 0,1,1,3 | 1,4,2,2 | 2,1,4,4 | 2,0,3,1
     | 0,1,2,2 | 1,5,1,2 | 1,0,3,5 | 4,1,1,4
     | 5,3,2,1 | 0,1,5,3 | 1,0,4,1 | 3,0,3,1
     | 0,1,2,1 | 2,1,5,3 | 2,4,4,1 | 4,3,1,3
     | 3,0,1,2 | 1,2,3,5 | 0,2,1,4 | 0,0,1,3
     | 1,0,3,4 | 0,3,3,1 | 0,0,4,3 | 1,0,3,1
     | 0,1,4,3 | 3,1,1,3 | 0,0,3,4 | 0,0,5,4
     | 0,0,4,5 |];
```



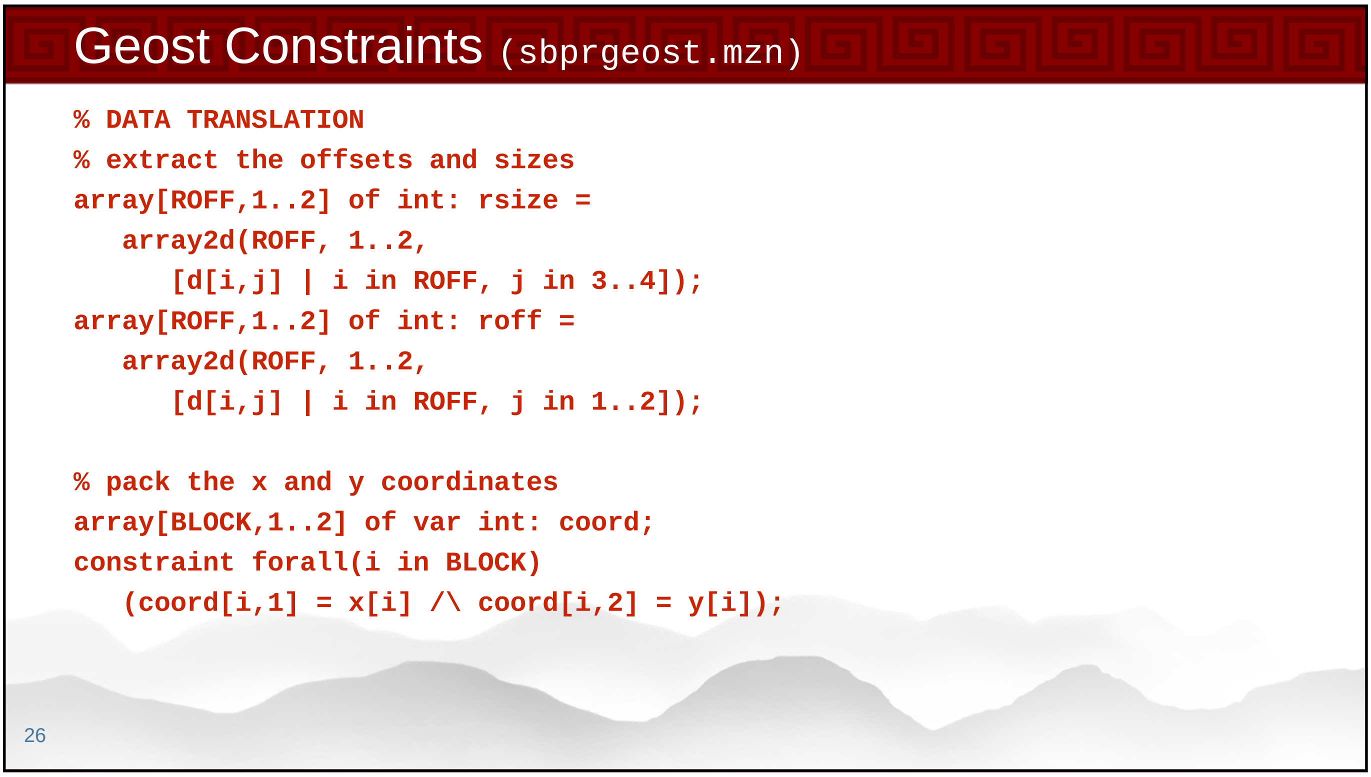
Geost Data File(sbprgeost.dzn)

```
% All shape+orientations are in a 1-d array
shape = [
  {1,2,3}, {3,4,5}, {6,4,7}, {8,2,9},
  {10,11,12}, {13,14,15}, {16,17,18}, {19,20,21},
  {17,22,23,24}, {11,25,26,27}, {28,29,30,31},
      {32,33,34,35},
  {44}, {45},
  {36,37}, {38,39}, {40,41}, {42,43}
];

% Define the shapes and the associated rotations
shapeind = [ {1,2,3,4}, {5,6,7,8}, {9,10,11,12},
  {13,14}, {15,16,17,18} ];
```



Geost Data, Decisions and Objective (sbprgeost.mzn)

```
int: n; % number of blocks
set of int: BLOCK = 1..n;
int: m; % number of rectangle/offsets
set of int: ROFF = 1..m;
array[ROFF,1..4] of int: d; % defns
array[int] of set of ROFF: shape;
int: h; % width of river
int: maxl; % maximum length of river

array[BLOCK] of var 0..maxl: x;
array[BLOCK] of var 0..h: y;
var 0..maxl: l; % length of river used

solve minimize l;
```



Geost Constraints (sbprgeost.mzn)

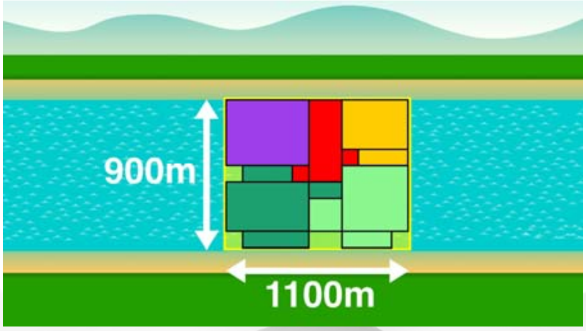
```
% DATA TRANSLATION
% extract the offsets and sizes
array[ROFF,1..2] of int: rsize =
   array2d(ROFF, 1..2,
      [d[i,j] | i in ROFF, j in 3..4]);
array[ROFF,1..2] of int: roff =
   array2d(ROFF, 1..2,
      [d[i,j] | i in ROFF, j in 1..2]);

% pack the x and y coordinates
array[BLOCK,1..2] of var int: coord;
constraint forall(i in BLOCK)
   (coord[i,1] = x[i] /\ coord[i,2] = y[i]);
```

Geost Constraints (sbprgeost.mzn)

```
% set up the "kind" constraints
array[BLOCK] of var int: kind;
array[BLOCK] of set of int: shapeind;
constraint forall(i in BLOCK)
   (kind[i] in shapeind[i]);
include "geost.mzn";
constraint geost_bb(2,
         rsize,
         roff,
         shape,
         coord,
         kind,
         [ 0,0 ],
         [ l,h ]);
```

Solving the Model

```
l = 11;
x = [4, 5, 0, 0, 7];
y = [4, 0, 0, 5, 5];
kind = [3, 8, 10, 13, 17];
----------
==========
Finished in 1s 179msec
```



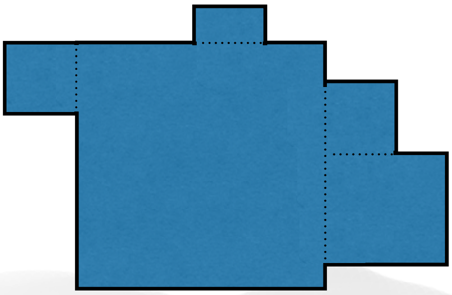
Carpet Cutting

- Carpeting a new house
  - measure each room size and shape
  - cut the carpets out of a roll of carpet of fixed width
  - lay the carpet
- Using the least length of the roll means
  - less wastage
  - more profit for the carpeting company
- Complexities
  - carpet direction
  - stairs, filler carpets, weave constraints

29

Carpet Orientation

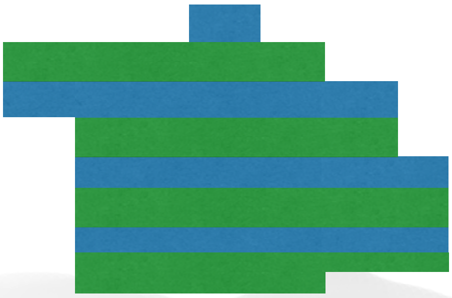
- On a uniform unpatterned carpet, we can cut a room in 4 different ways



30

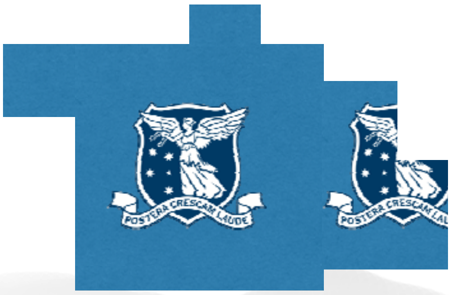
Carpet Orientation

- On a horizontally striped carpet we can cut a room in 2 different ways



Carpet Orientation

- On a non-symmetric patterned carpet we can cut a room in only 1 way



Summary

- Complex packing problems
  - make shapes from components
  - ensure components don't overlap
  - rotations/orientations add to the complexity of the problem
- Globals
  - `diffn_k` (for *k* dimensional packing)
  - `geost` (for flexible *k* dimensional packing)
- In practice most packing is 2D or 3D

Image Credits

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