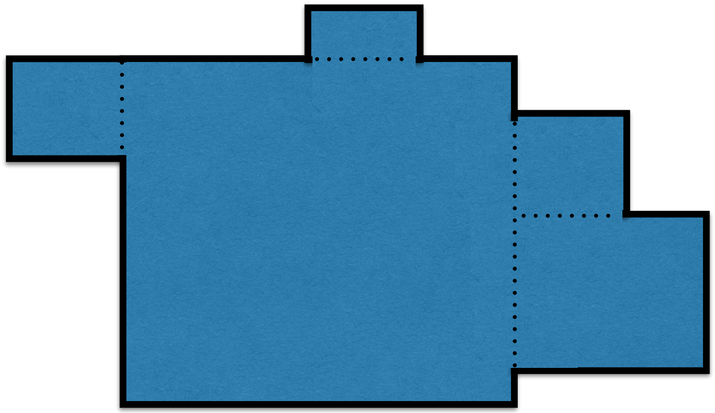
Rectilinear Packing without Rotation

Peter Stuckey & Guido Tack

MONASH University

Complex Block Packing

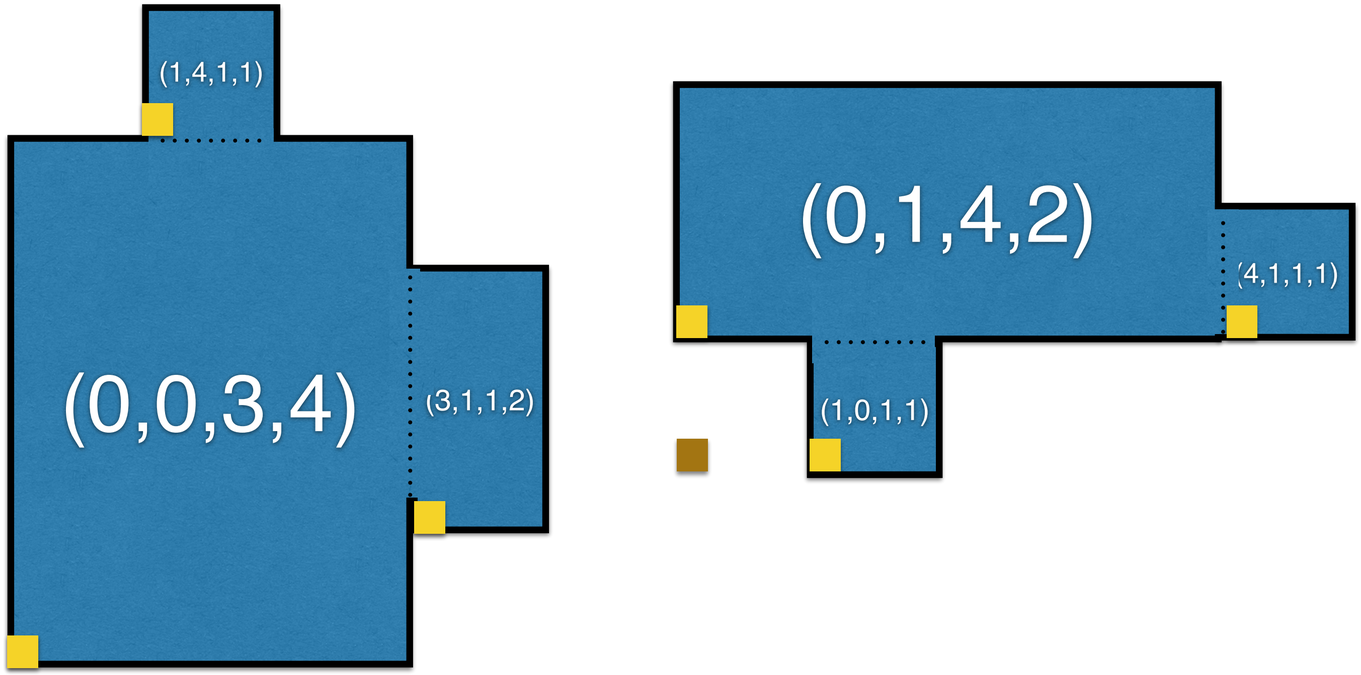
- Given a set of **rectilinear** shapes



- Pack them together so that
  - the resulting height/width does not exceed `h`
  - the resulting width `w` is minimized

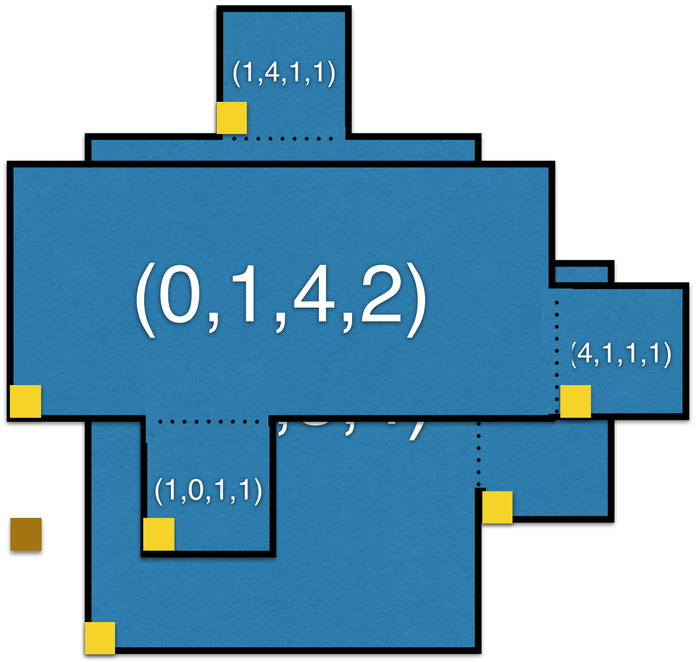
Representing Block Shapes

- Rectangles at offset to shape bottom left
  - $(x_{offset}, y_{offset}, x_{size}, y_{size})$
- The offsets are **different** when the orientation is different (more later)



Representing Block Shapes

- Represent the component rectangles with offsets
- A block (of a specific orientation) is a set of rectangles with offsets
- Component rectangles can be shared if possible
- E.g. a list of components
  - 1: 0,0,3,4
  - 2: 0,1,4,2
  - 3: 1,4,1,1
  - 4: 3,1,1,2
  - 5: 4,1,1,1
  - 6: 1,0,1,1



representation {1,3,4}

Block Packing Data

- Given `n` blocks defined by fixed shapes. Place the blocks in a rectangle of height `h` so they don't overlap and minimise the width `w`

```
enum Block;
enum Comp;
enum Dims = {XOff, YOff, XSize, YSize};
array[Comp, Dims] of int: dims;
array[Block] of set of Comp: shape;

int: h;    % height of rectangle
int: maxw; % maximum width of rectangle
```

Block Packing Data

```
Block = B(1..5);
Comp = C(1..12);
dims = [| 1,0,2,5 % (x_offset,y_offset,x_size,y_size)
        | 3,4,1,1
        | 0,3,1,1
        | 1,4,2,2
        | 0,1,1,3 % shared by blocks 2 & 3
        | 1,0,4,4
        | 1,5,1,2
        | 1,0,3,5
        | 4,1,1,4
        | 0,0,1,3
        | 1,0,3,4
        | 0,0,4,5 |];
shape = [ {C(1),C(2),C(3)}, {C(4),C(5),C(6)}, {C(5),C(7),C(8),C(9)},
          {C(12)}, {C(10),C(11)} ];
h = 9; maxw = 16;
```

Block Packing Decisions + Objective

- For each block
  - `x` position of its base
  - `y` position of its base

```
array[Block] of var 0..maxw: x;
array[Block] of var 0..h: y;

var 0..maxw: w; % width of rectangle used


solve minimize w;
```

Block Packing Constraints

- Each rectangle component must fit within the full area

```
forall(i in Block)(
  forall(r in shape[i])(
    x[i]+d[r,XOff]+d[r,XSize] <= w /\
    y[i]+d[r,YOff]+d[r,YSize] <= h
  )
);
```

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

Block Packing Constraints

- Rectangle/offsets don't overlap

```
forall(i,j in Block where i < j)(
forall(r1 in shape[i],r2 in shape[j])(
   x[i]+dims[r1,XOff]+dims[r1,XSize]
                  <= x[j]+dims[r2,XOff]
\/ x[j]+dims[r2,XOff]+dims[r2,XSize]
                  <= x[i]+dims[r1,XOff]
\/ y[i]+dims[r1,YOff]+dims[r1,YSize]
                  <= y[j]+dims[r2,YOff]
\/ y[j]+dims[r2,YOff]+dims[r2,YSize]
                  <= y[i]+dims[r1,YOff]
  )
);
```

Block Packing Constraints

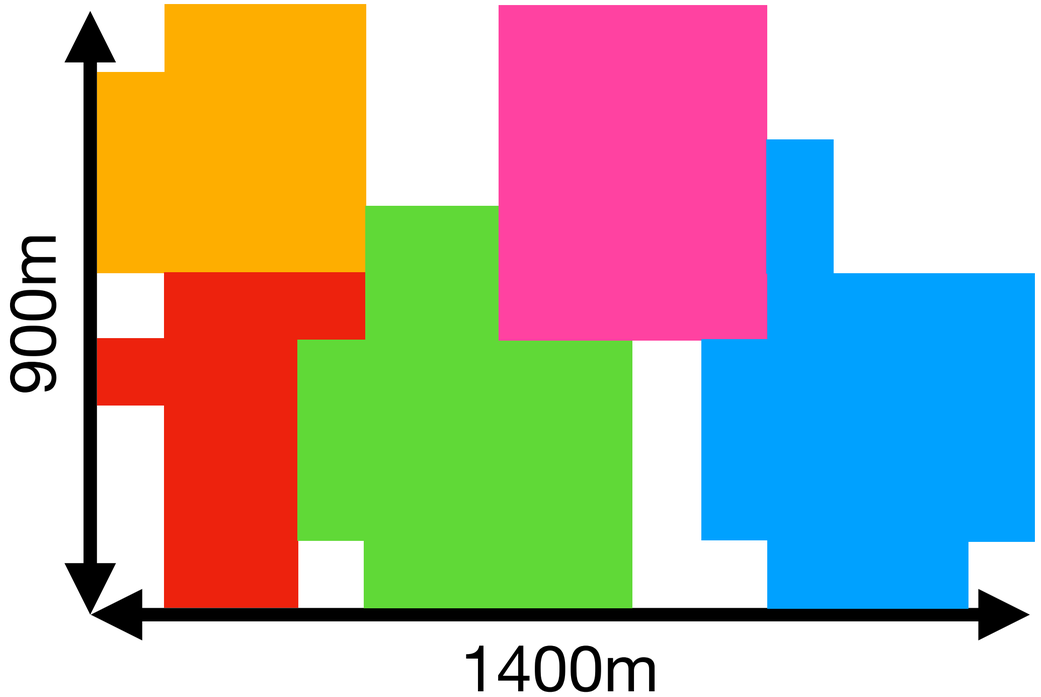
- We can once again use the `diffn` global constraint

```
diffn(
   [x[b]+dims[s,XOff]|b in Block,s in shape[b]],
   [y[b]+dims[s,YOff]|b in Block,s in shape[b]],
   [dims[s,XSize]|b in Block,s in shape[b]],
   [dims[s,YSize]|b in Block,s in shape[b]],
);
```

- **Note**: the global constraint `diffn` is extensible to *k* dimensions
  - in MiniZinc `diffn_k`

Solving the Model

```
l = 14;
x = [0, 3, 9, 6, 0];
y = [0, 0, 0, 4, 5]
----------
==========
```



Summary

- Complex packing problems
  - make shapes from components
  - ensure components don't overlap
- Packing bulk mineral cargoes onto storage pads at ports is an example of complex rectilinear packing (without rotation)
- Problem gets even more complicated when orientation/rotation is taken into account!

EOF