

Discrete Optimization

Constraint Programming: Part IV

Goal of the lecture

- ▶ Illustrating the modeling techniques in constraint programming
 - symmetry breaking

Symmetries

- ▶ Many problems naturally exhibit symmetries
 - exploring symmetrical parts of the search space is useless

Symmetries

- ▶ Many problems naturally exhibit symmetries
 - exploring symmetrical parts of the search space is useless
- ▶ Many kinds of symmetries
 - variable symmetries
 - value symmetries

Symmetries

- ▶ Many problems naturally exhibit symmetries
 - exploring symmetrical parts of the search space is useless
- ▶ Many kinds of symmetries
 - variable symmetries
 - value symmetries
- ▶ The next slides
 - symmetry-breaking constraints

BIBDs

- ▶ **Balanced Incomplete Block Designs (BIBDs)**
 - Input: (v, b, r, k, l)
 - Output: a v by b 0/1 matrix with exactly r ones per row, k ones per column, and a scalar product of value l
- ▶ **Why BIBDs?**
 - example of combinatorial design
 - full of variable symmetries

BIBDs

- ▶ Balanced Incomplete Block Designs (BIBDs)

- Input: (v, b, r, k, l)
- Output: a v by b 0/1 matrix with exactly r ones per row, k ones per column, and a scalar product of value l

- ▶ Why BIBDs?

- example of combinatorial design
- full of variable symmetries

$(3, 3, 2, 2, 1)$

1	1	0
0	1	1
1	0	1

BIBDs

- ▶ **Balanced Incomplete Block Designs (BIBDs)**

- Input: (v, b, r, k, l)
- Output: a v by b 0/1 matrix with exactly r ones per row, k ones per column, and a scalar product of value l

- ▶ **Why BIBDs?**

- example of combinatorial design
- full of variable symmetries

$(3, 3, 2, 2, 1)$

1	1	0
0	1	1
1	0	1

BIBDs

```
range Rows = 1..v;  
range Cols = 1..b;  
var{int} m[Rows,Cols] in 0..1;  
solve {  
    forall(i in Rows)  
        sum(y in Cols) m[i,y] = r;  
    forall(j in Cols)  
        sum(x in Rows) m[x,j] = k;  
    forall(i in Rows, j in Rows: j > i)  
        sum(x in Cols) (m[i,x] & m[j,x]) = 1;  
}
```

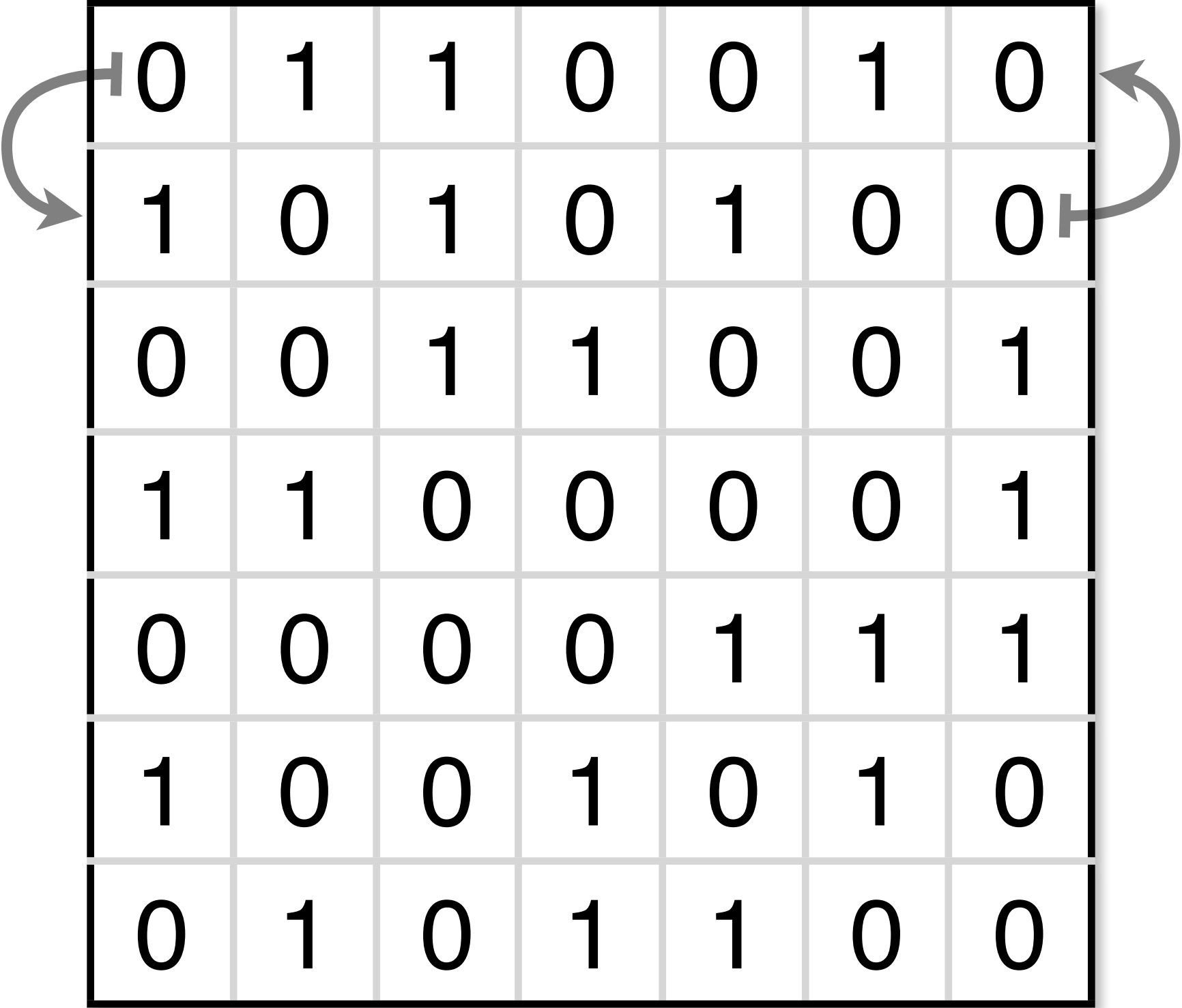
BIBDs

$(7,7,3,3,1)$

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

BIBDs

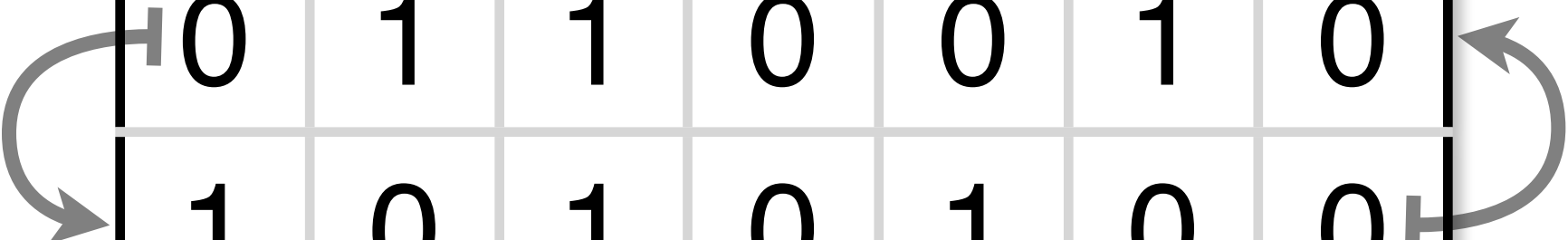
$(7,7,3,3,1)$



0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

BIBDs

$(7,7,3,3,1)$



0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

$(7,7,3,3,1)$

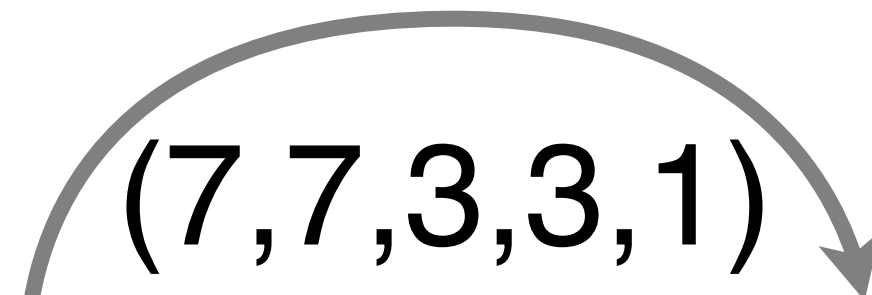
1	0	1	0	1	0	0
0	1	1	0	0	1	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

BIBDs

$(7,7,3,3,1)$

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

BIBDs



(7, 7, 3, 3, 1)

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

BIBDs

(7,7,3,3,1)

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

(7,7,3,3,1)

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	0	0	0	0	1	1
0	1	0	0	1	0	1
1	1	0	1	0	0	0
0	0	0	1	1	1	0

BIBDs

- ▶ How to break variable symmetries
 - impose an ordering on the variables

BIBDs

- ▶ How to break variable symmetries
 - impose an ordering on the variables
- ▶ Consider the row symmetries
 - impose a lexicographic constraint

BIBDs

- ▶ How to break variable symmetries
 - impose an ordering on the variables
- ▶ Consider the row symmetries
 - impose a lexicographic constraint
- ▶ Lexicographic ordering
 - a: 0 1 1 0 0 1 0 1 1 1 0 0 1 0
 - b: 1 0 1 0 1 0 0 1 0 1 0 1 0 0
 - $a \leq b$ $a \geq b$

BIBDs

$(7,7,3,3,1)$

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

BIBDs

(7,7,3,3,1)

0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	0	1	1	0	0	1
1	1	0	0	0	0	1
0	0	0	0	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	0	0

Lexicographic Ordering

(7,7,3,3,1)

0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	0	0	1

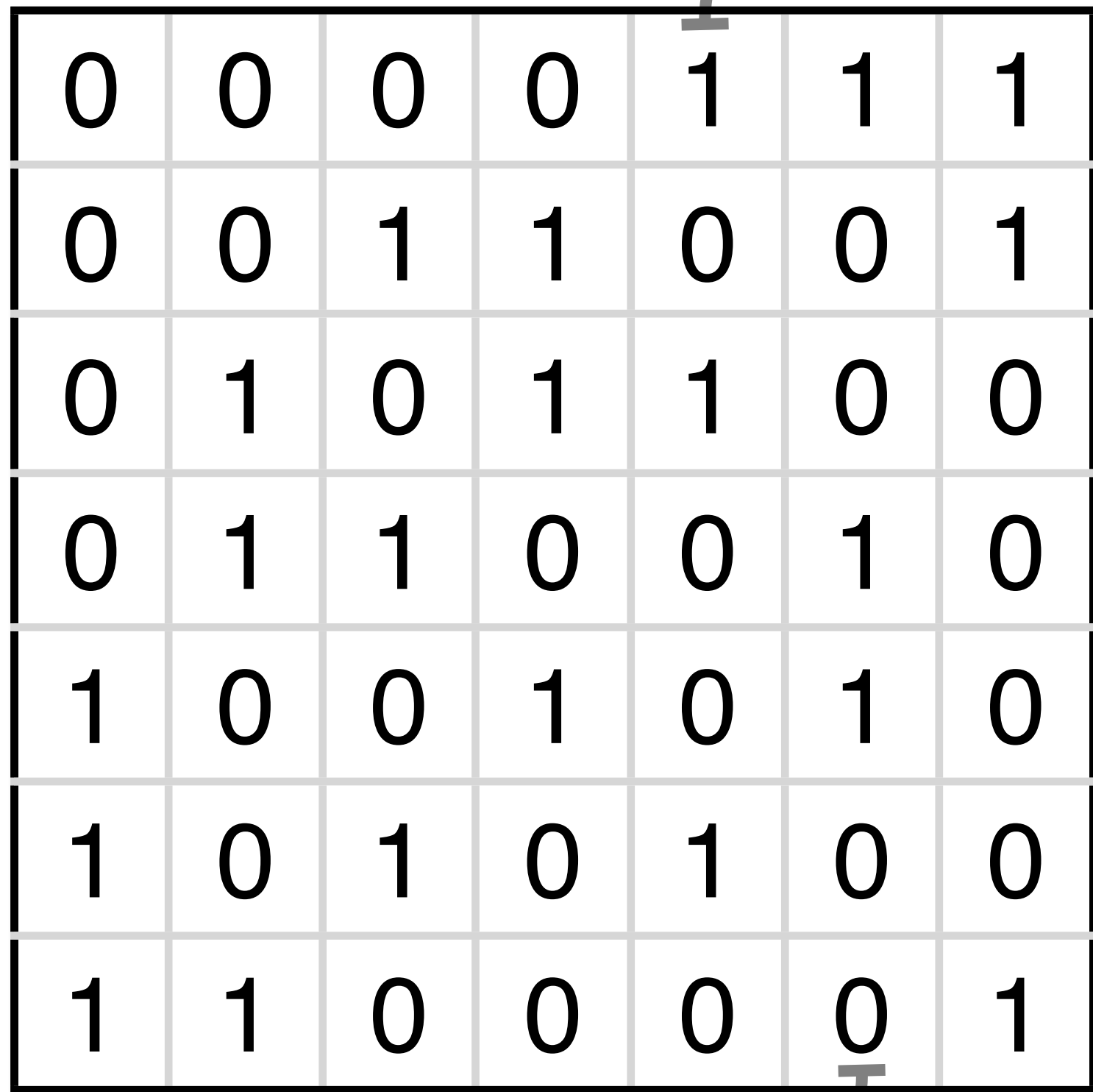
BIBDs

$(7,7,3,3,1)$

0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	0	0	1

BIBDs

$(7, 7, 3, 3, 1)$



0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	0	0	1

BIBDs

(7,7,3,3,1)

0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	0	0	1

(7,7,3,3,1)

0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	1

BIBDs

```
range Rows = 1..v;
range Cols = 1..b;
var{int} m[Rows,Cols] in 0..1;
solve {
    forall(i in Rows)
        sum(y in Cols) m[i,y] = r;
    forall(j in Cols)
        sum(x in Rows) m[x,j] = k;
    forall(i in Rows, j in Rows: j > i)
        sum(x in Cols) (m[i,x] & m[j,x]) = 1;
    forall(i in 1..v-1)
        lexleq(all(j in Cols) m[i,j], all(j in Cols) m[i+1,j]);
    forall(j in 1..b-1)
        lexleq(all(i in Rows) m[i,j], all(i in Rows) m[i,j+1]);
}
```

Scene Allocation

```
range Scenes = ...;
range Days   = ...;
range Actor  = ...;
int fee[Actor] = ...;
set{Actor} appears[Scenes] = ...;
set{int} which[a in Actor] = setof(i in Scenes) member(a, appears[i]);
var{int} shoot[Scenes] in Days;

minimize
    sum(a in Actor) sum(d in Days)
        fee[a] * or(s in which[a]) (shoot[s]=d)
subject to
    atmost(all(i in Days) 5, Days, shoot);
```

Scene Allocation

- ▶ Value symmetries

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable
 - I can swap all the scenes in day 1 and all the scenes in day 2 and I still have a solution

Scene Allocation

- ▶ **Value symmetries**
 - the days are interchangeable
 - I can swap all the scenes in day 1 and all the scenes in day 2 and I still have a solution
 - nothing can be used to distinguish the days: they are interchangeable

Scene Allocation

► Value symmetries

- the days are interchangeable
- I can swap all the scenes in day 1 and all the scenes in day 2 and I still have a solution
- nothing can be used to distinguish the days: they are interchangeable
- if s is a solution, then $p(s)$ is a solution where the days of s have been permuted by permutation p

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable
 - I can swap all the scenes in day 1 and all the scenes in day 2 and I still have a solution
 - nothing can be used to distinguish the days: they are interchangeable
 - if s is a solution, then $p(s)$ is a solution where the days of s have been permuted by permutation p
- ▶ How can we eliminate these symmetries?

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable
 - I can swap all the scenes in day 1 and all the scenes in day 2 and I still have a solution
 - nothing can be used to distinguish the days: they are interchangeable
 - if s is a solution, then $p(s)$ is a solution where the days of s have been permuted by permutation p
- ▶ How can we eliminate these symmetries?
 - Consider the first scene s_1 . What are the days that we consider for this scene?

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable
 - if s is a solution, then $p(s)$ is a solution where the days of s have been permuted by permutation p
- ▶ How can we eliminate these symmetries?
 - consider the first scene s_1 . What are the days that we consider for this scene?
 - consider the second scene s_2 . Which days must be considered for s_2 ?

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable
 - if s is a solution, then $p(s)$ is a solution where the days of s have been permuted by permutation p
- ▶ How can we eliminate these symmetries?
 - consider the first scene s_1 . What are the days that we consider for this scene?

Only one day, say day 1
 - consider the second scene s_2 . Which days must be considered for s_2 ?

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable
 - if s is a solution, then $p(s)$ is a solution where the days of s have been permuted by permutation p
- ▶ How can we eliminate these symmetries?
 - consider the first scene s_1 . What are the days that we consider for this scene?

Only one day, say day 1

– cons
days

All days are interchangeable
at this point

ch

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable
 - if s is a solution, then $p(s)$ is a solution where the days of s have been permuted by permutation p
- ▶ How can we eliminate these symmetries?
 - consider the first scene s_1 . What are the days that we consider for this scene?

Only one day, say day 1
 - consider the second scene s_2 . Which days must be considered for s_2 ?

Scene Allocation

- ▶ Value symmetries
 - the days are interchangeable
 - if s is a solution, then $p(s)$ is a solution where the days of s have been permuted by permutation p
- ▶ How can we eliminate these symmetries?
 - consider the first scene s_1 . What are the days that we consider for this scene?

Only one day, say day 1

- consider the second scene s_2 . Which days must be considered for s_2 ?

Only two days, day 1 and day 2

Scene Allocation

► How can we eliminate these symmetries?

- consider the first scene s_1 . What are the days that we consider for this scene?

Only one day, say day 1

- consider the second scene s_2 . Which days must be considered for s_2 ?

Only two days, day 1 and day 2

Scene Allocation

- ▶ How can we eliminate these symmetries?
 - consider the first scene s_1 . What are the days that we consider for this scene?

Only one day, say day 1
 - consider the second scene s_2 . Which days must be considered for s_2 ?

Only two days, day 1 and day 2
 - In general the already used days and one additional new day

Scene Allocation

► How can we eliminate these symmetries?

- consider the first scene s_1 . What are the days that we consider for this scene?

Only one day, say day 1

- consider the second scene s_2 . Which days must be considered for s_2 ?

Only two days, day 1 and day 2

- In general the already used days and one additional new day

existing days

new day

$$D(s_k) = \{1, 2, \dots, \max(s_1, \dots, s_{k-1}) + 1\}$$

Scene Allocation

```
range Scenes = 1..n;
range Days   = 1..m;
range Actor  = ...;
int fee[Actor] = ...;
set{Actor} appears[Scenes] = ...;
set{int} which[a in Actor] = setof(i in Scenes) member(a,appears[i]);
var{int} shoot[Scenes] in Days;

minimize
    sum(a in Actor) sum(d in Days)
        fee[a] * or(s in which[a]) (shoot[s]=d)
subject to {
    atmost(all(i in Days) 5,Days,shoot);
    scene[1] = 1;
    forall(s in Scenes: s > 1)
        scene[s] <= max(k in 1..s-1) scene[k] + 1;
}
```