Value Symmetries

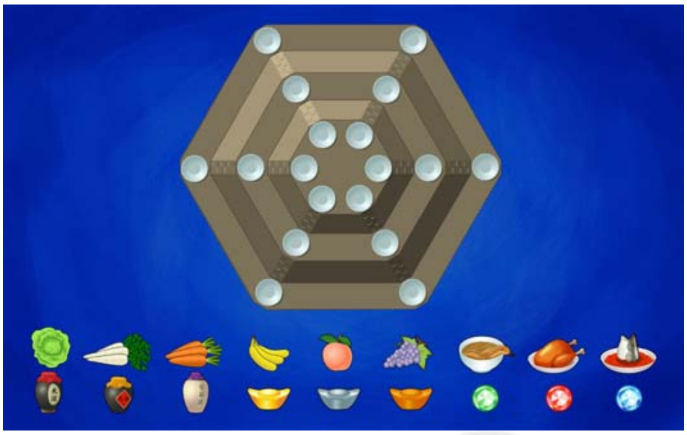
Jimmy Lee & Peter Stuckey

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

Decorating an Altar

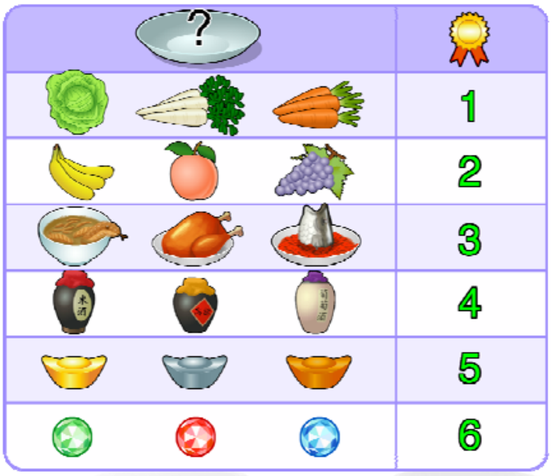
2

Decorating an Altar



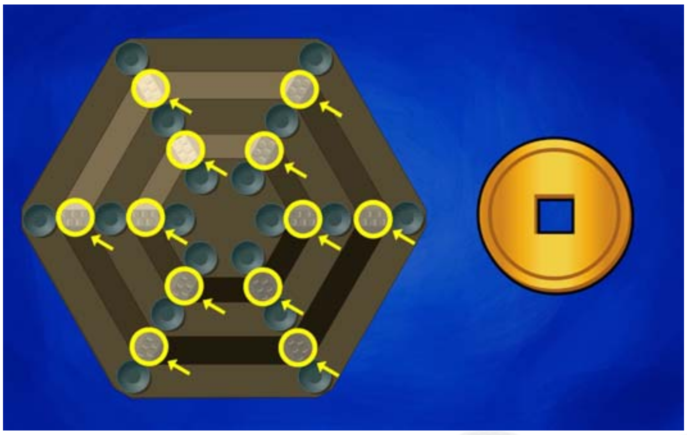
3

Ranks of Tributes

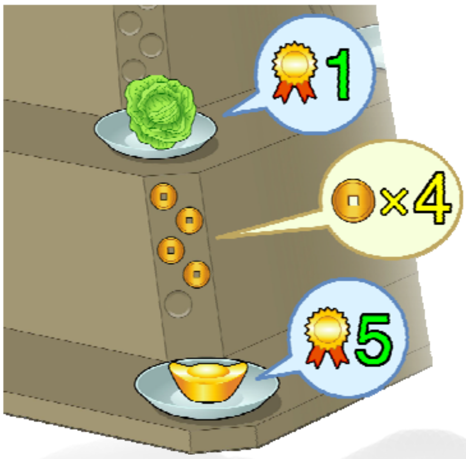


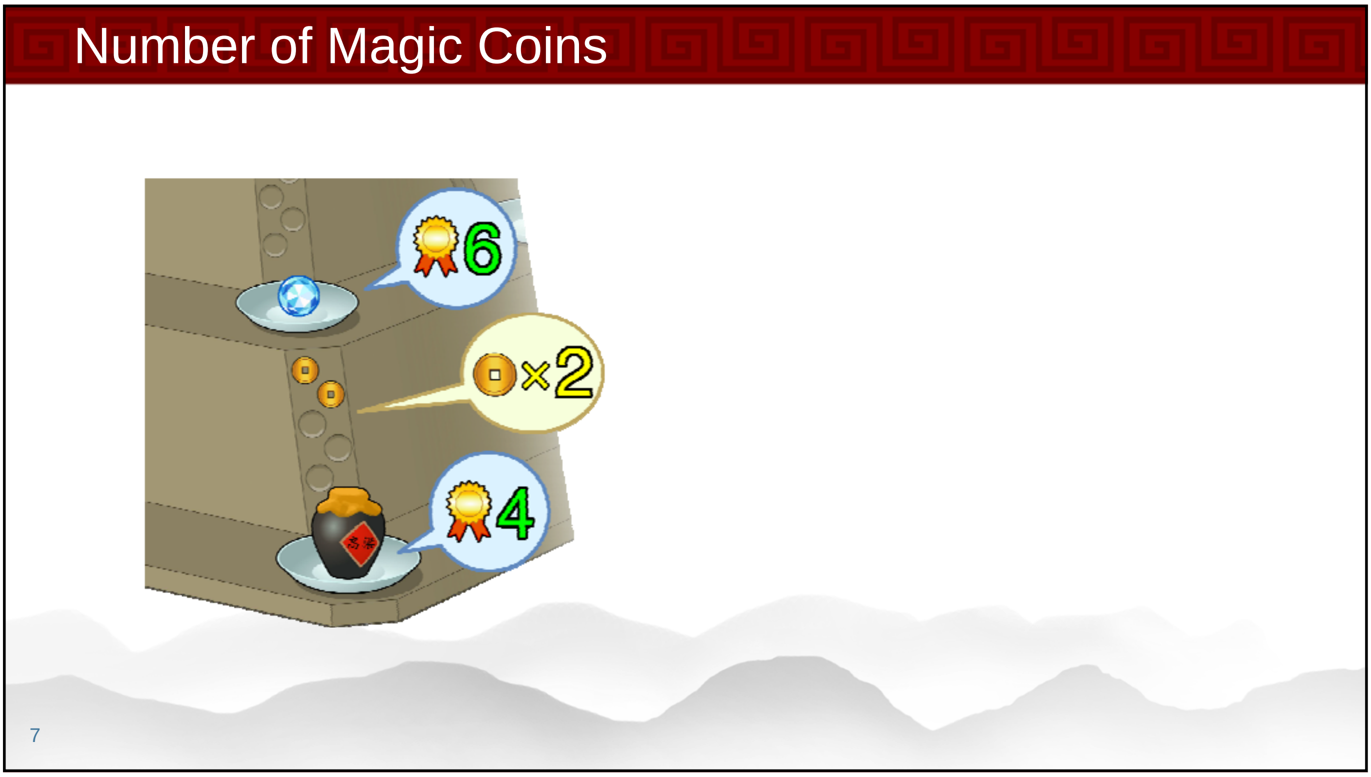
4

Magic Coins



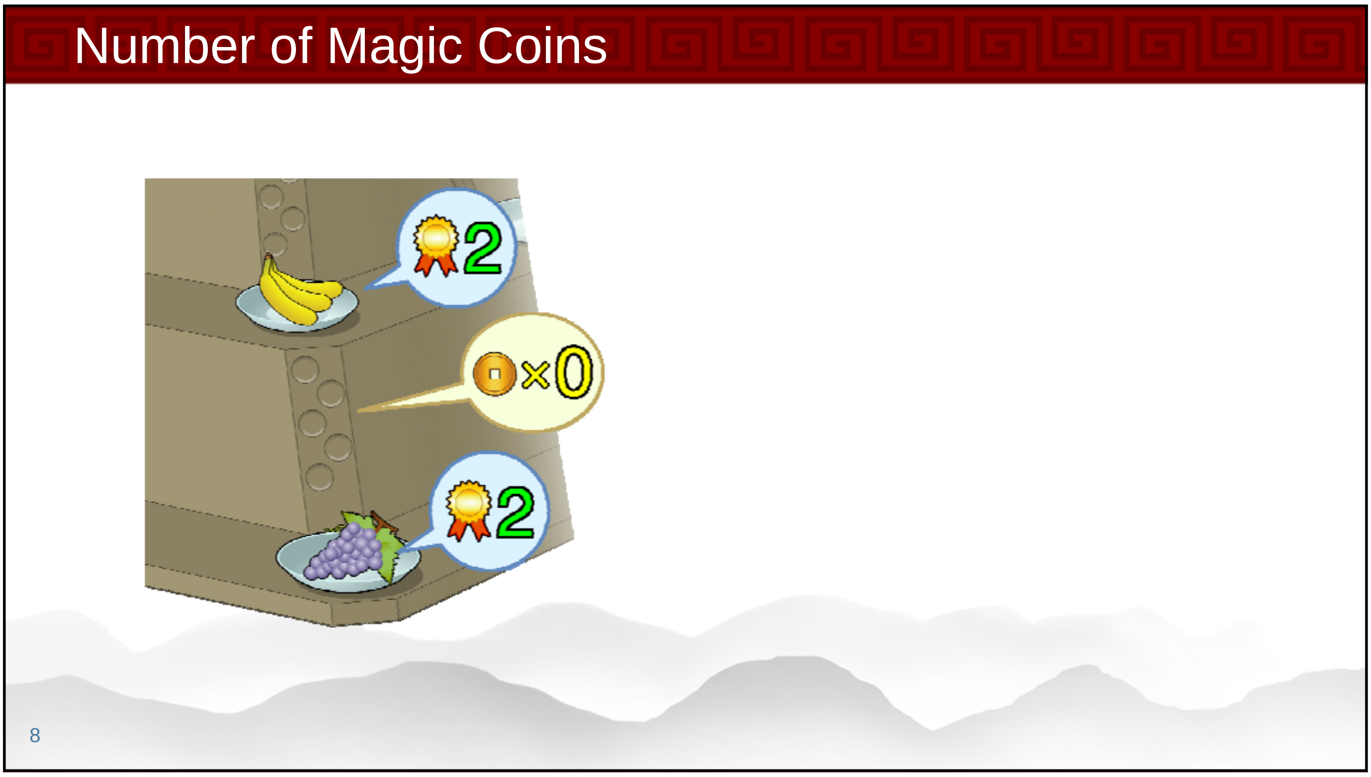
Number of Magic Coins





Number of Magic Coins

7



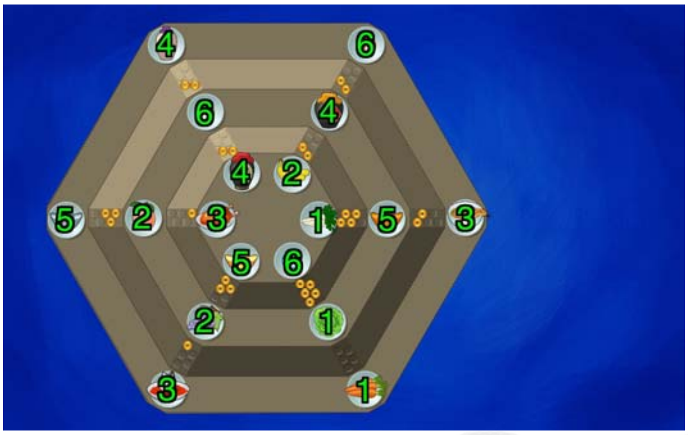
Number of Magic Coins

8

Towards a Divine Arrangement

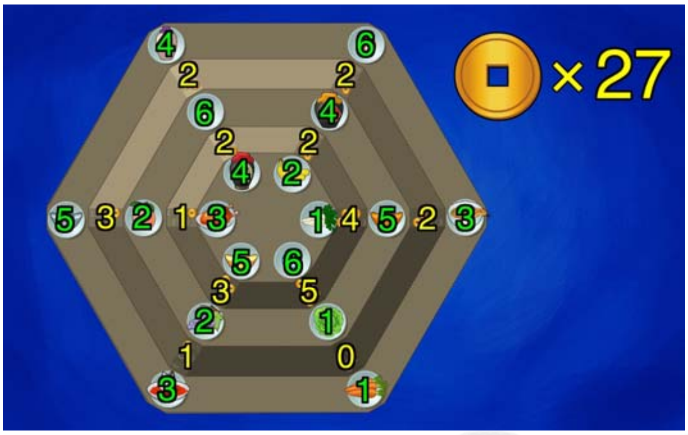
9

Towards a Divine Arrangement



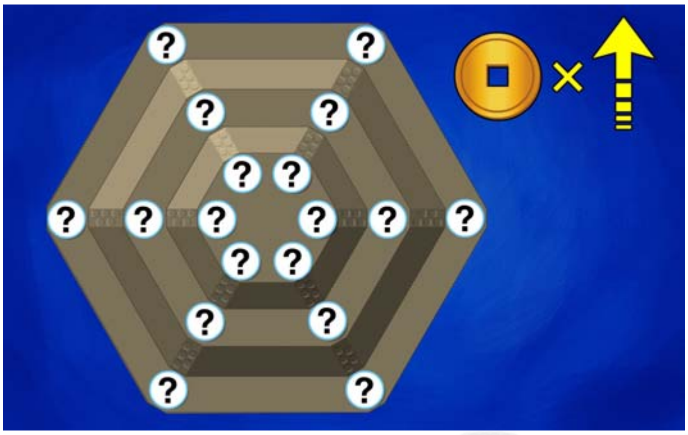
10

Towards a Divine Arrangement

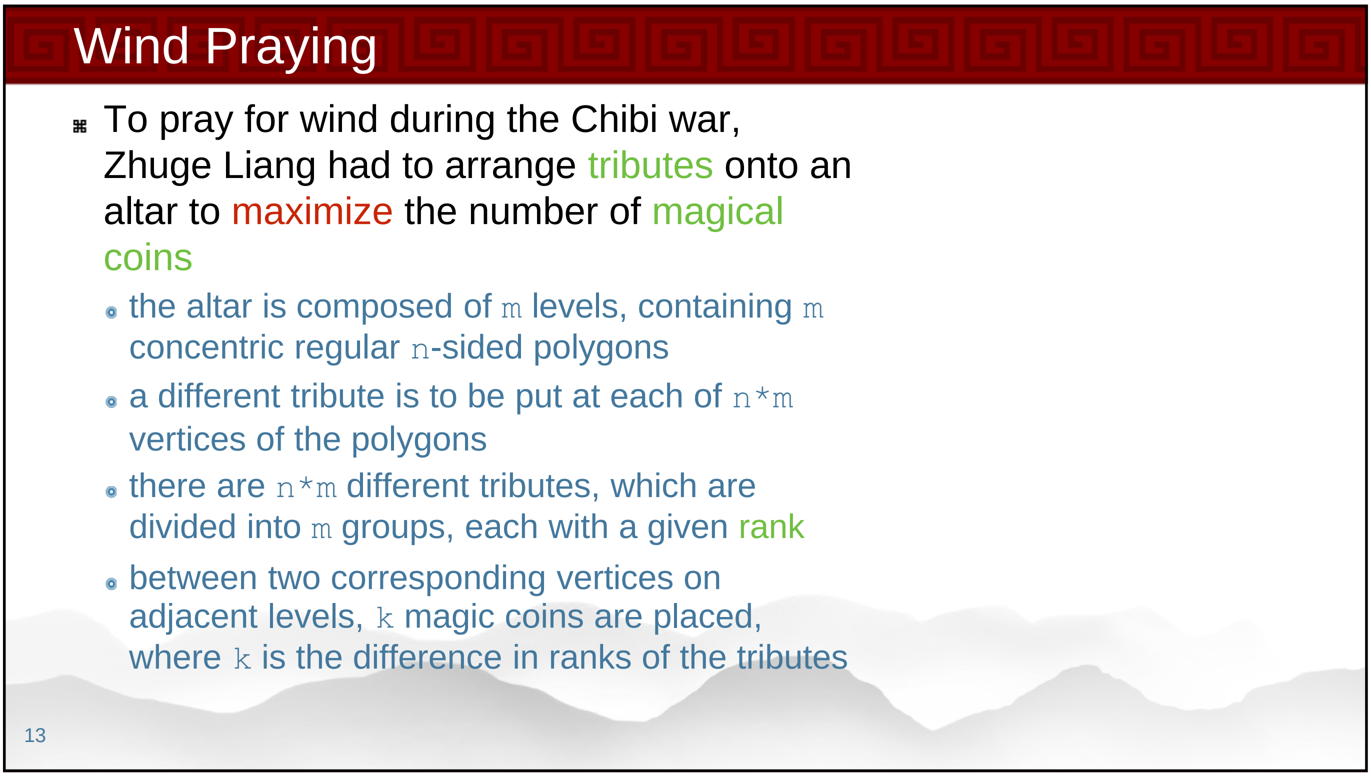


11

Towards a Divine Arrangement

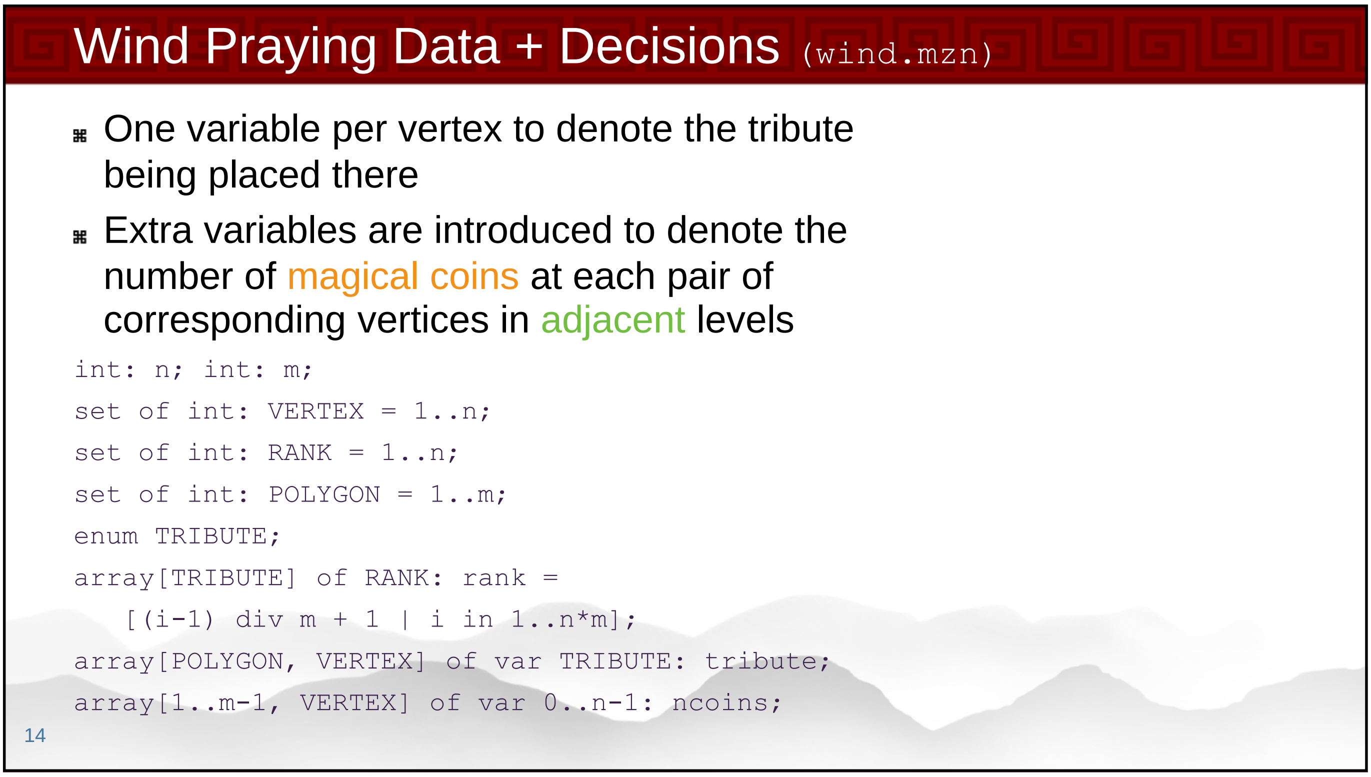


12



Wind Praying

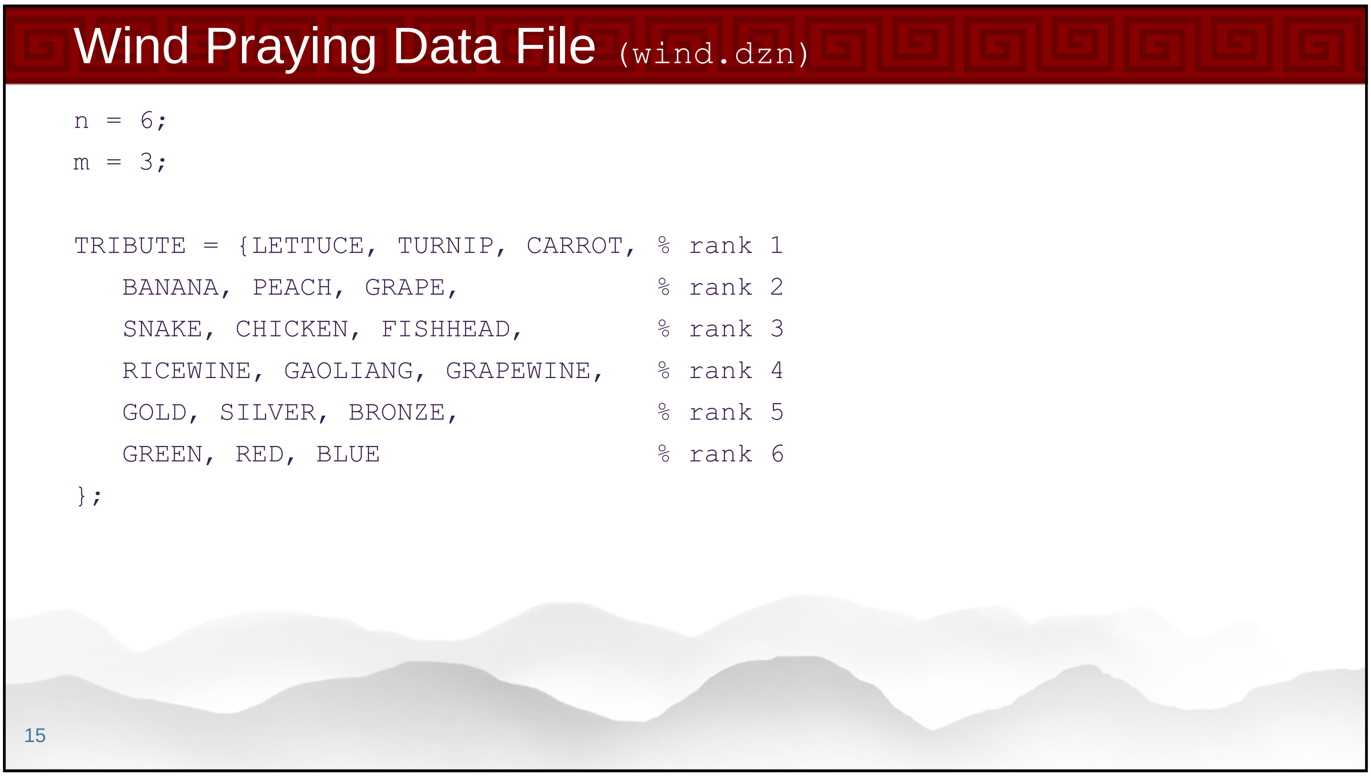
- To pray for wind during the Chibi war, Zhuge Liang had to arrange tributes onto an altar to maximize the number of magical coins
  - the altar is composed of `m` levels, containing `m` concentric regular `n`-sided polygons
  - a different tribute is to be put at each of `n*m` vertices of the polygons
  - there are `n*m` different tributes, which are divided into `m` groups, each with a given rank
  - between two corresponding vertices on adjacent levels, `k` magic coins are placed, where `k` is the difference in ranks of the tributes



Wind Praying Data + Decisions (wind.mzn)

- One variable per vertex to denote the tribute being placed there
- Extra variables are introduced to denote the number of magical coins at each pair of corresponding vertices in adjacent levels

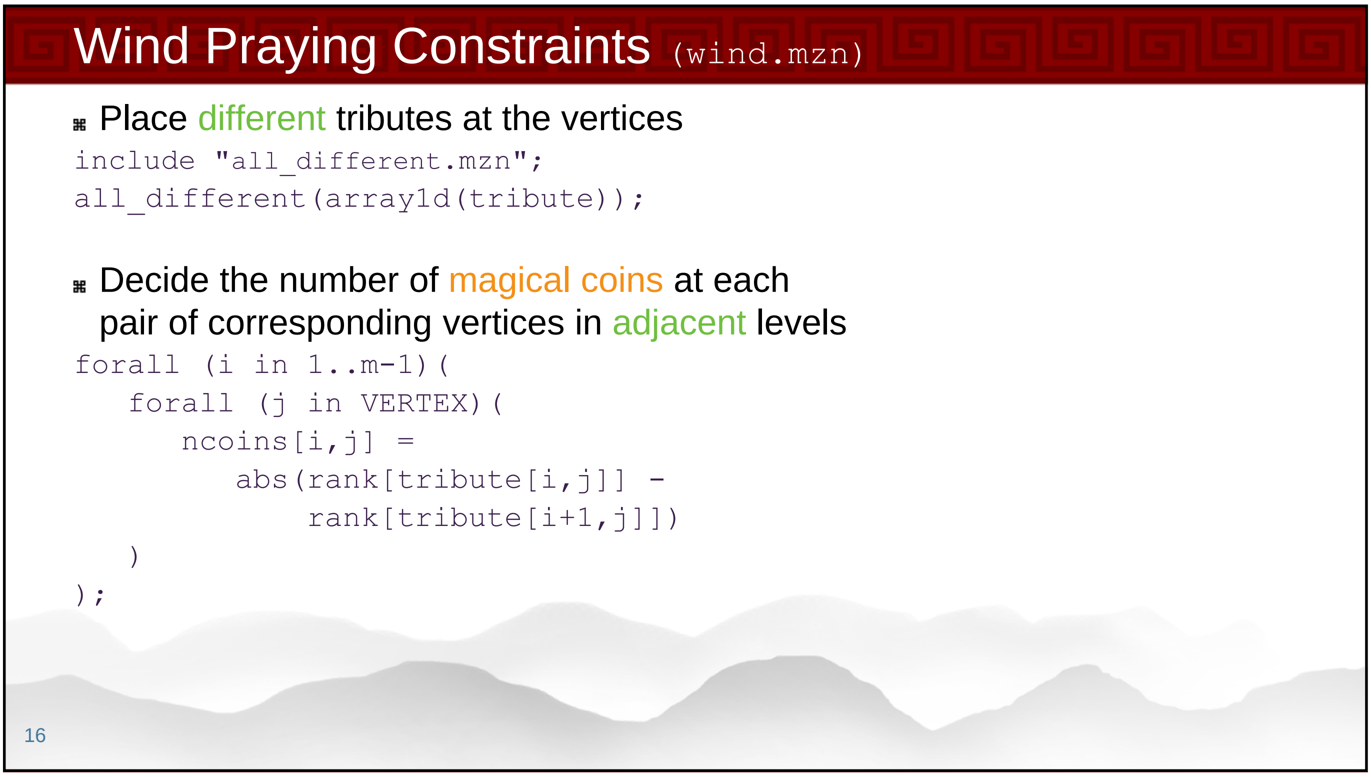
```
int: n; int: m;
set of int: VERTEX = 1..n;
set of int: RANK = 1..n;
set of int: POLYGON = 1..m;
enum TRIBUTE;
array[TRIBUTE] of RANK: rank =
   [(i-1) div m + 1 | i in 1..n*m];
array[POLYGON, VERTEX] of var TRIBUTE: tribute;
array[1..m-1, VERTEX] of var 0..n-1: ncoins;
```



Wind Praying Data File (wind.dzn)

```
n = 6;
m = 3;

TRIBUTE = {LETTUCE, TURNIP, CARROT, % rank 1
   BANANA, PEACH, GRAPE,            % rank 2
   SNAKE, CHICKEN, FISHHEAD,        % rank 3
   RICEWINE, GAOLIANG, GRAPEWINE,   % rank 4
   GOLD, SILVER, BRONZE,            % rank 5
   GREEN, RED, BLUE                 % rank 6
};
```



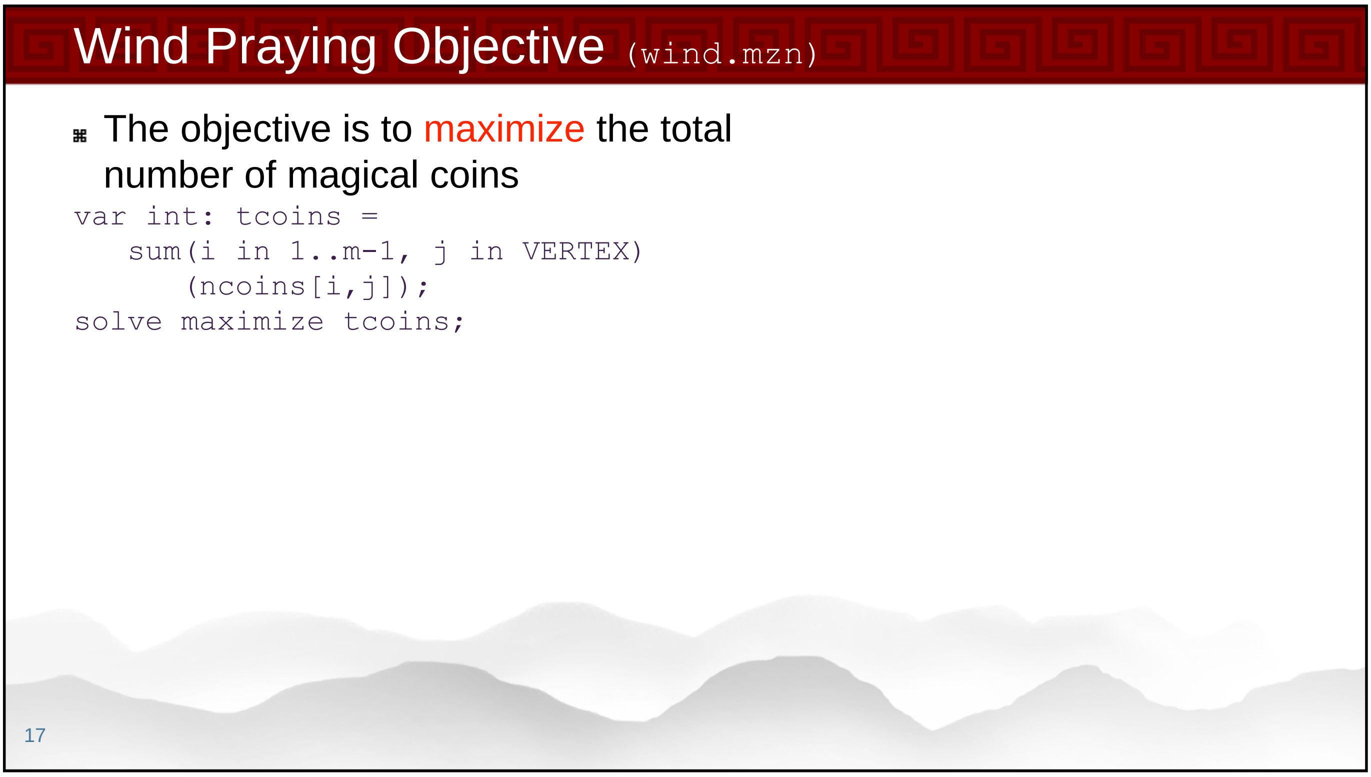
Wind Praying Constraints (wind.mzn)

- Place different tributes at the vertices

```
include "all_different.mzn";
all_different(array1d(tribute));
```

- Decide the number of magical coins at each pair of corresponding vertices in adjacent levels

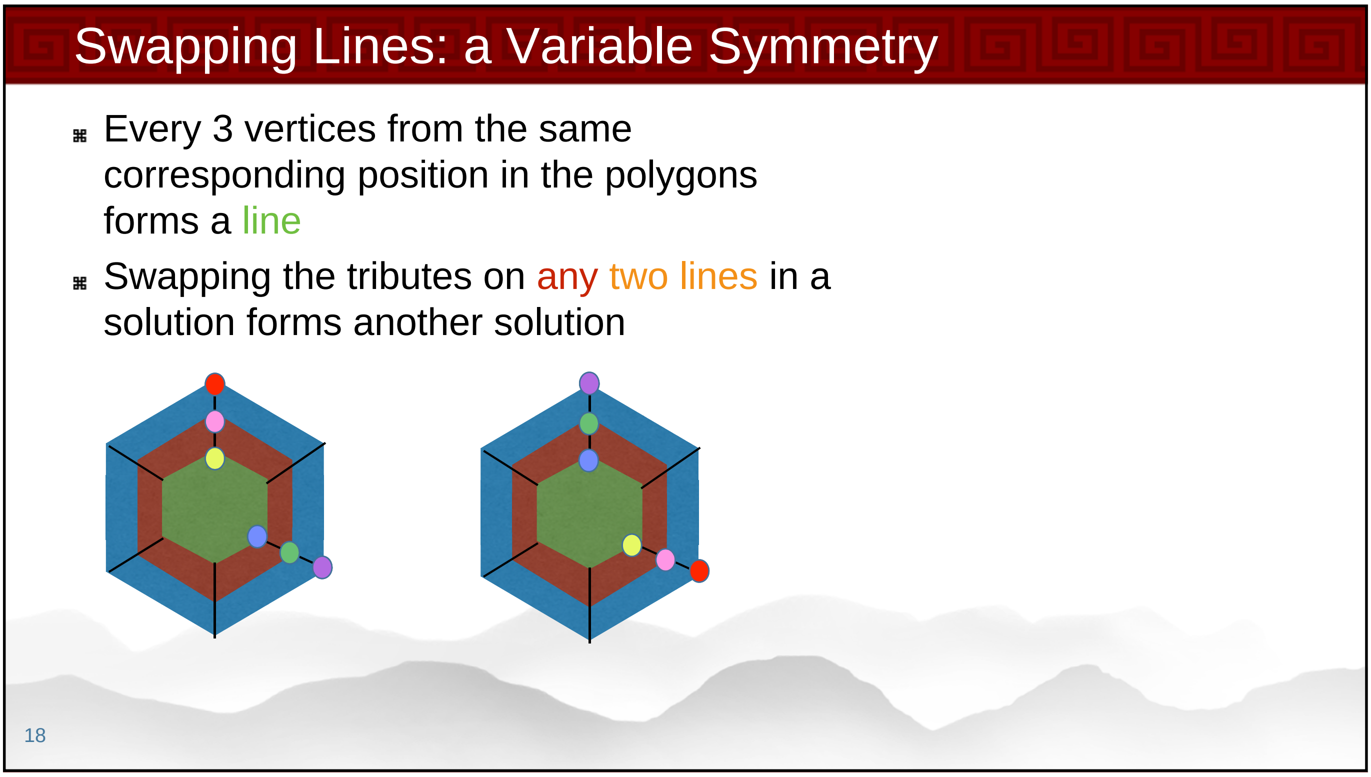
```
forall (i in 1..m-1)(
   forall (j in VERTEX)(
      ncoins[i,j] =
         abs(rank[tribute[i,j]] -
             rank[tribute[i+1,j]])
   )
);
```



Wind Praying Objective (wind.mzn)

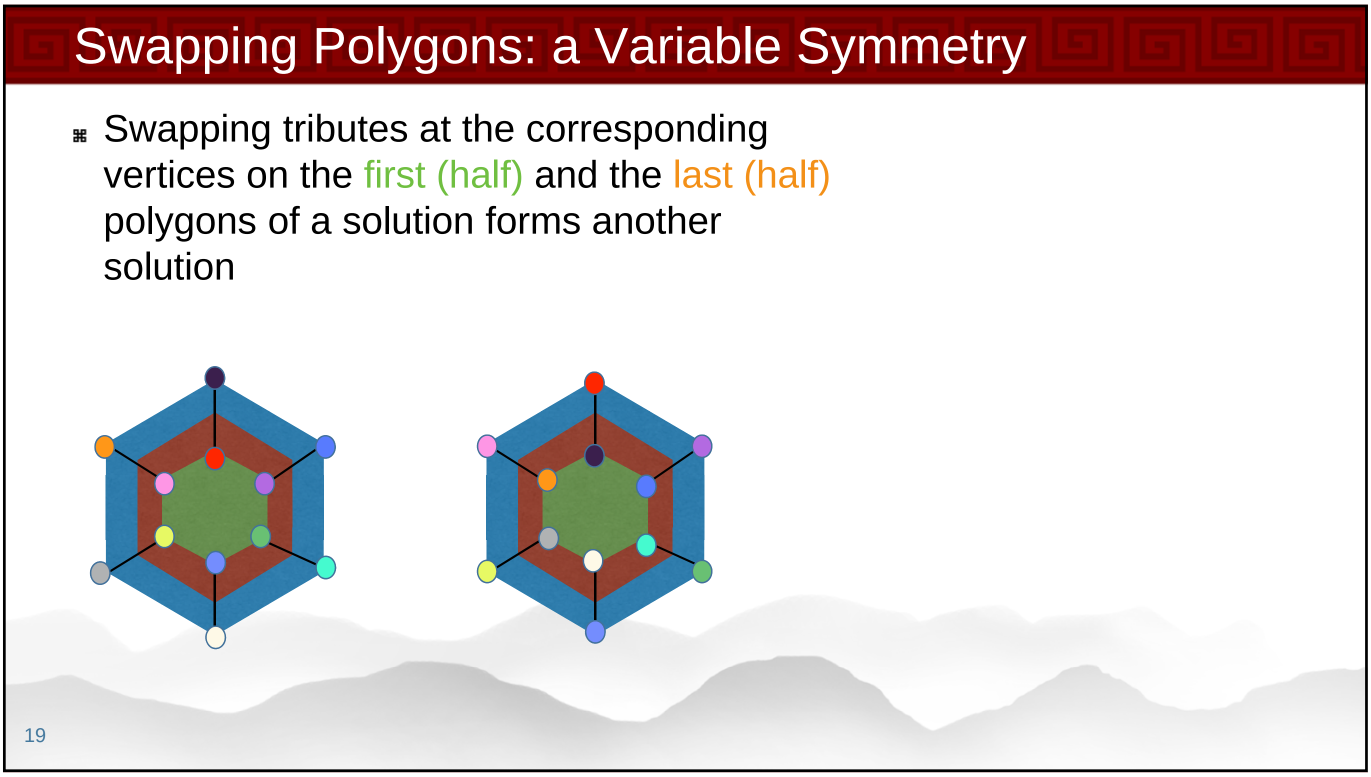
- The objective is to maximize the total number of magical coins

```
var int: tcoins =
   sum(i in 1..m-1, j in VERTEX)
      (ncoins[i,j]);
solve maximize tcoins;
```



Swapping Lines: a Variable Symmetry

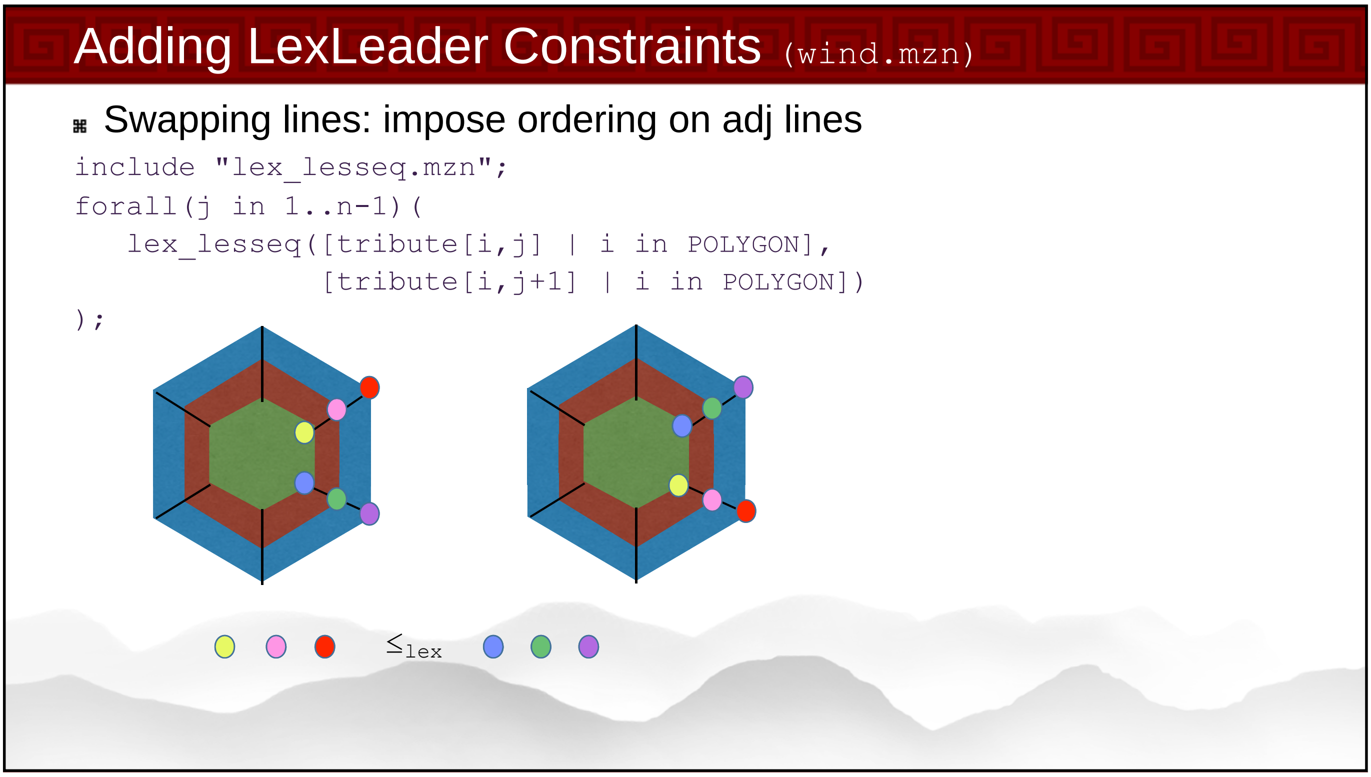
- Every 3 vertices from the same corresponding position in the polygons forms a line
- Swapping the tributes on any two lines in a solution forms another solution



Swapping Polygons: a Variable Symmetry

- Swapping tributes at the corresponding vertices on the first (half) and the last (half) polygons of a solution forms another solution

19



Adding LexLeader Constraints (wind.mzn)

- Swapping lines: impose ordering on adj lines

```
include "lex_lesseq.mzn";
forall(j in 1..n-1)(
   lex_lesseq([tribute[i,j] | i in POLYGON],
              [tribute[i,j+1] | i in POLYGON])
);
```

● ● ● $\leq_{lex}$ ● ● ●

Adding LexLeader Constraints (wind.mzn)

- Swapping lines: impose ordering on adj lines

```
include "lex_lesseq.mzn";
forall(j in 1..n-1)(
   lex_lesseq([tribute[i,j] | i in POLYGON],
              [tribute[i,j+1] | i in POLYGON])
);
```

● ● ● $\leq_{lex}$ ● ● ●

- Since tributes are all different (● must be different from ●), these can be simplified to:

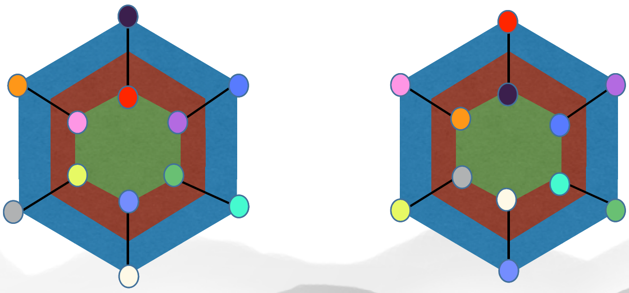
```
forall(j in 1..n-1)
      (tribute[1,j] < tribute[1,j+1]);
```

● < ●

Adding LexLeader Constraints (wind.mzn)

- Swapping the first (half) and last (half) polygons: impose ordering on polygons

```
lex_lesseq([tribute[i,j] |
   i in 1..round(m div 2), j in VERTEX],
           [tribute[m+1-i,j] |
   i in 1..round(m div 2), j in VERTEX]);
```



● ● ● ● ● ● $\leq_{lex}$ ● ● ● ● ● ●

Adding LexLeader Constraints (wind.mzn)

- Swapping the first (half) and last (half) polygons: impose ordering on polygons

```
lex_lesseq([tribute[i,j] |
   i in 1..round(m div 2), j in VERTEX],
           [tribute[m+1-i,j] |
   i in 1..round(m div 2), j in VERTEX]);
```

● ● ● ● ● ● $\leq_{lex}$ ● ● ○ ● ● ●

- Since tributes are all different, this can be simplified to

```
tribute[1,1] < tribute[m,1];
```

● < ●

Solving the Model

- Only 12 coins after 10m of solving
- 23 coins after 20m of solving

Value Symmetries

- A value symmetry g is a bijection on values that preserves solutions:

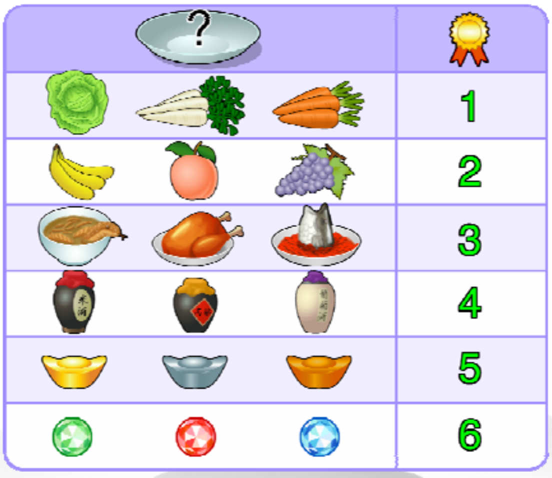
  $[x_1=v_1, \dots, x_n=v_n]$ is a solution

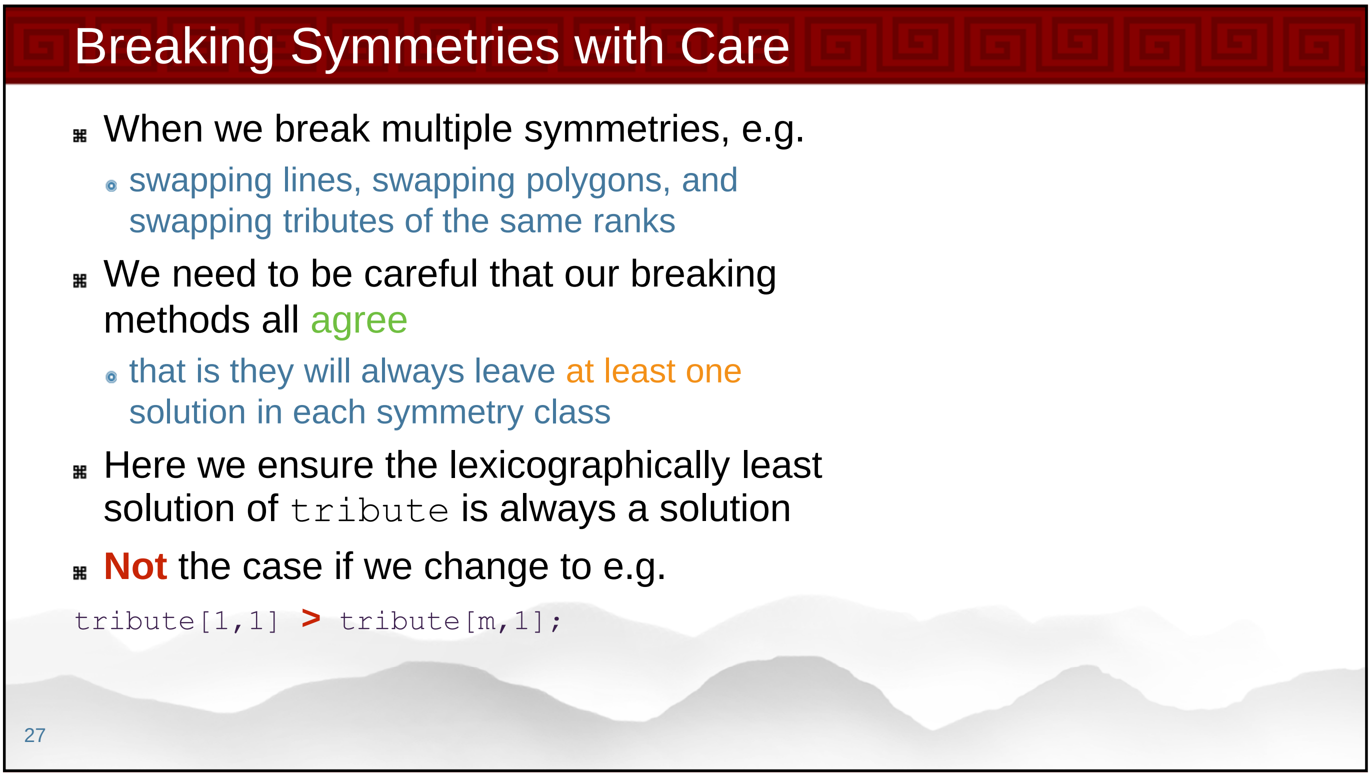
  $\Leftrightarrow$

  $[x_1=g(v_1), \dots, x_n=g(v_n)]$ is a solution
- We have already seen examples in
  - `catapult.mzn`: (clustering)
    - the names of clusters are all irrelevant
  - `table_seating_imp.mzn`: (seating scholars at a banquet)
    - the numbers of the tables are all irrelevant

Value Symmetries

- Tributes with the same ranks are interchangeable



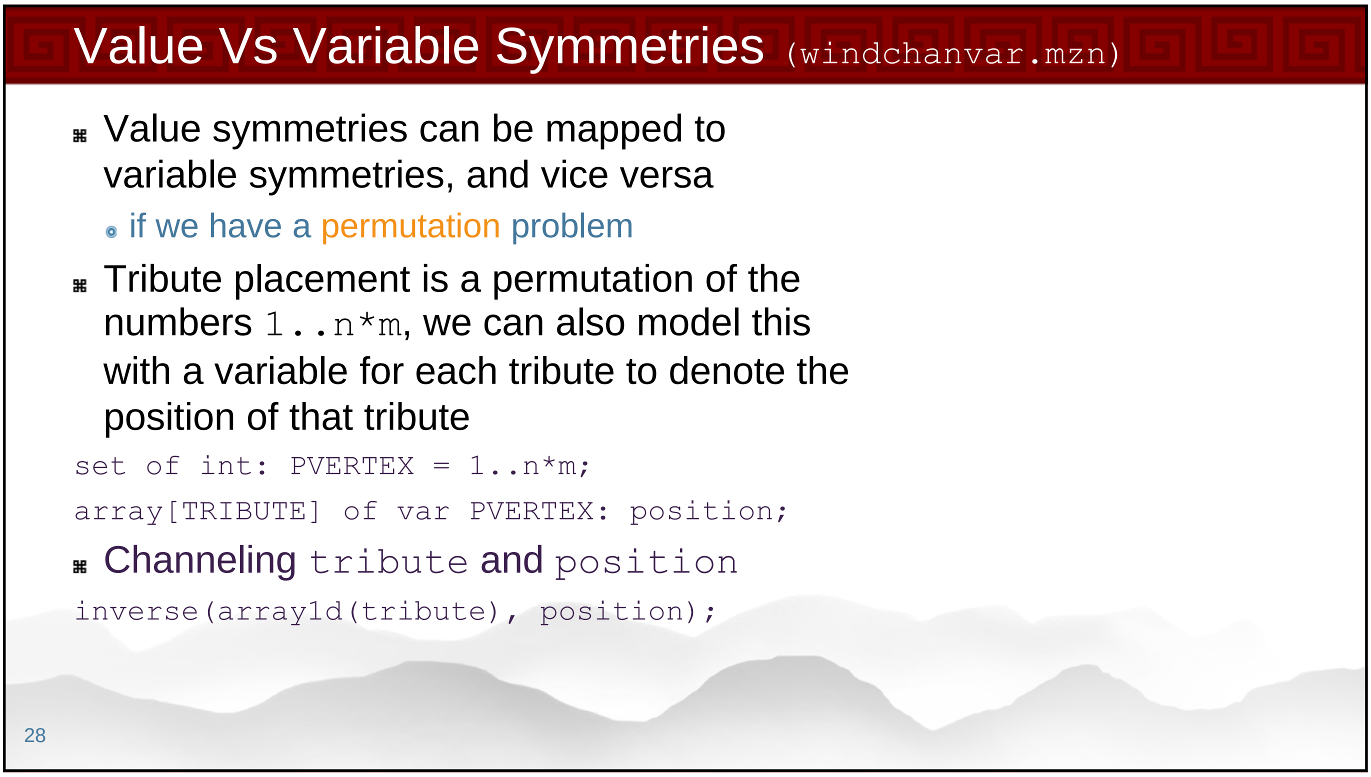


Breaking Symmetries with Care

- When we break multiple symmetries, e.g.
  - swapping lines, swapping polygons, and swapping tributes of the same ranks
- We need to be careful that our breaking methods all agree
  - that is they will always leave at least one solution in each symmetry class
- Here we ensure the lexicographically least solution of `tribute` is always a solution
- **Not** the case if we change to e.g.

```
tribute[1,1] > tribute[m,1];
```

27



Value Vs Variable Symmetries (windchanvar.mzn)

- Value symmetries can be mapped to variable symmetries, and vice versa
  - if we have a permutation problem
- Tribute placement is a permutation of the numbers `1..n*m`, we can also model this with a variable for each tribute to denote the position of that tribute

```
set of int: PVERTEX = 1..n*m;
array[TRIBUTE] of var PVERTEX: position;
```

- Channeling `tribute` and `position`

```
inverse(array1d(tribute), position);
```

28

From Value to Variable Symmetries

- Variables `position[i]` and `position[j]` are interchangeable if tributes `i` and `j` are of the same rank

- For example, if in one solution
  - `position[`🟢`] = 5`
  - `position[`🔴`] = 11`
  - `position[`🔵`] = 8`

- Permuting 🟢, 🔴 and 🔵 among themselves in the 3 assignments above gives another solution

Breaking Variable Symmetries (windchanvar.mzn)

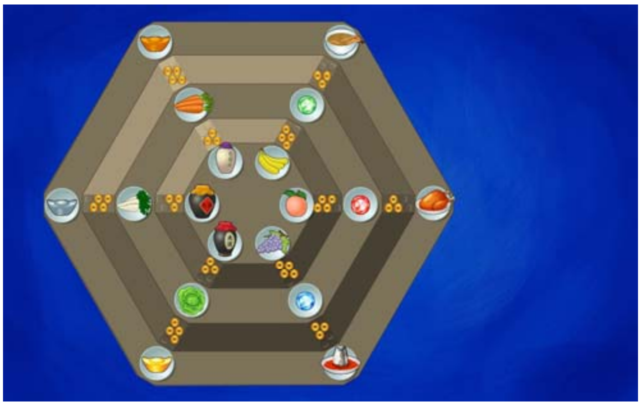
- Value symmetries in the original viewpoint are variable symmetries in the second viewpoint

- LexLeader constraints on interchangeable variables can be simplified as follows
  - actually imposing an order on `position[]` variables for tributes of the same rank

```
forall (i in RANK) (
   forall (j in m*(i-1)+1..m*i-1) (
      position[j] < position[j+1]
   )
);
```

Solving the Improved Model

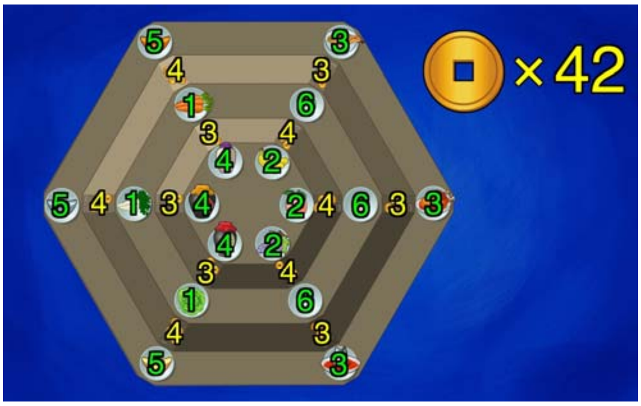
- Solution in 2m 26s



31

Solving the Improved Model

- Solution in 2m 26s



32

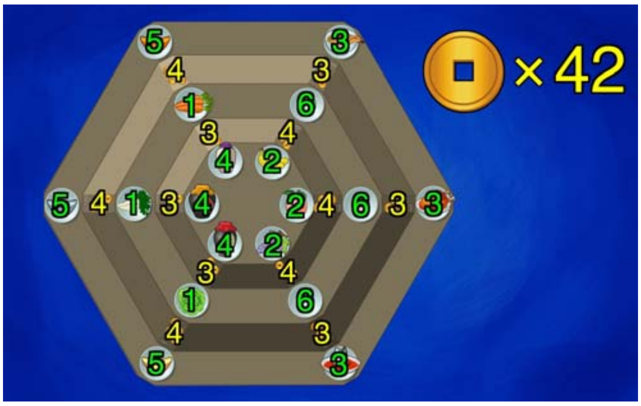
Removing Value Symmetries (windprec.mzn)

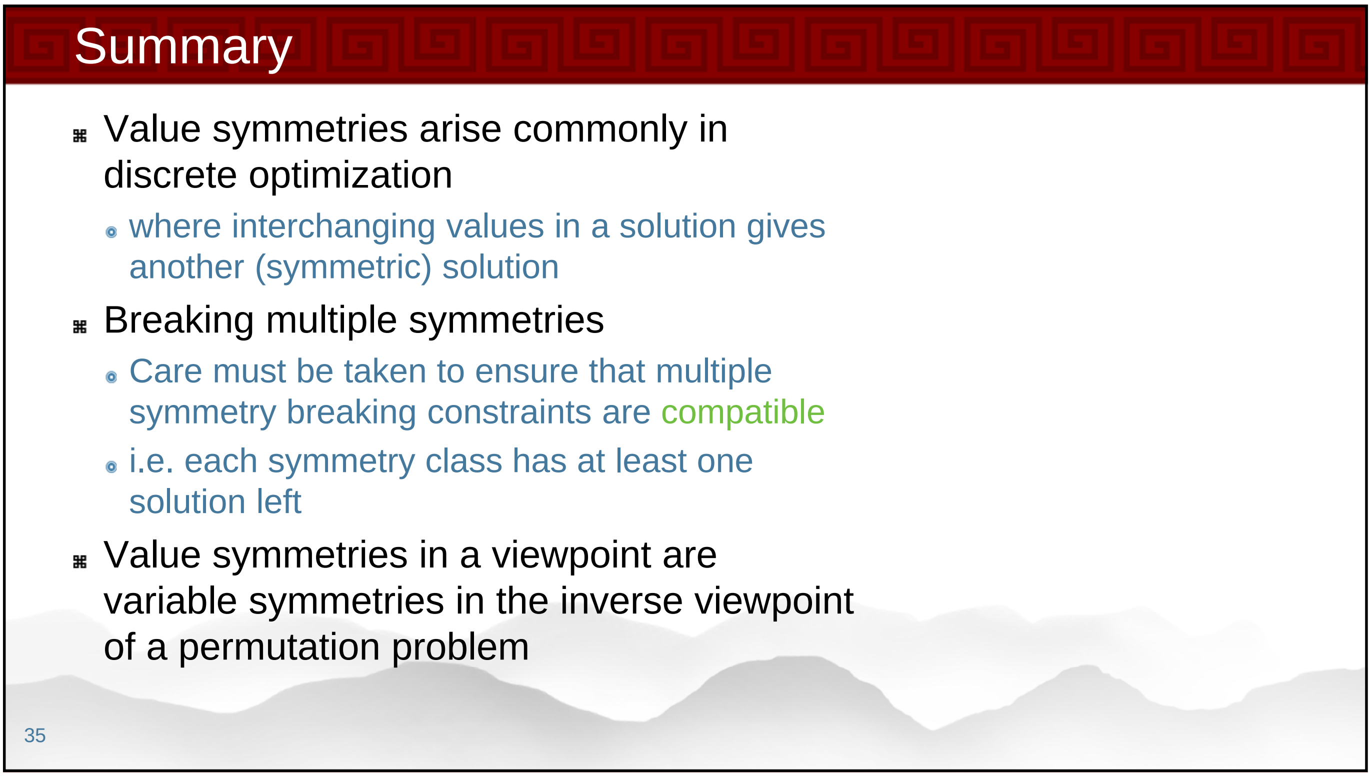
- Of course, we can use the global `value_precede_chain` in the original viewpoint to remove these symmetries too

```
include "value_precede_chain.mzn";
forall(r in RANK)(
   value_precede_chain(
      [i | i in (r-1)*m+1..r*m],
         array1d(tribute)));
```

Solving the Model with Precedence

- Solution in 15s





Summary

- Value symmetries arise commonly in discrete optimization
  - where interchanging values in a solution gives another (symmetric) solution
- Breaking multiple symmetries
  - Care must be taken to ensure that multiple symmetry breaking constraints are compatible
  - i.e. each symmetry class has at least one solution left
- Value symmetries in a viewpoint are variable symmetries in the inverse viewpoint of a permutation problem

35

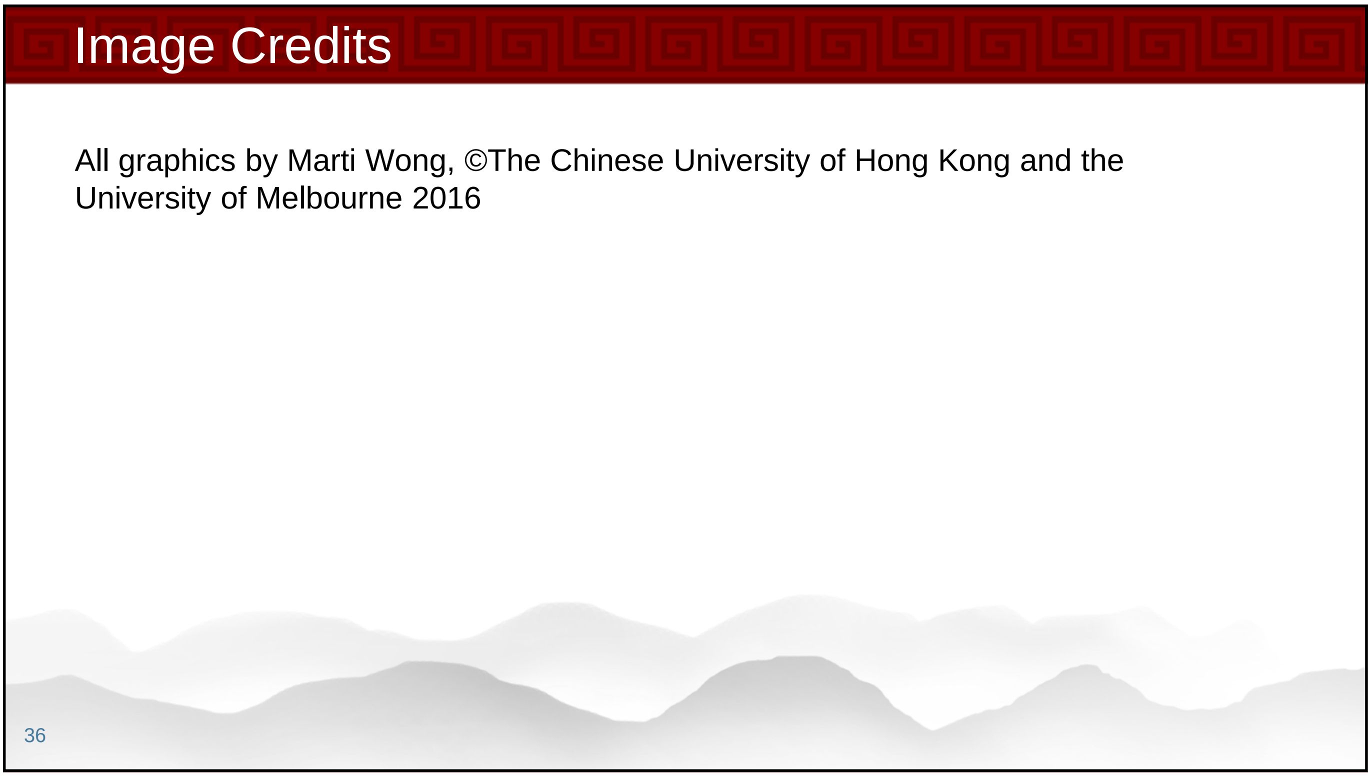


Image Credits

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

36