DM841

Constraint Programming

Symmetry Breaking

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Resume and Outlook

- Modeling in CP
- Global constraints (declaration)
- Notions of local consistency
- Global constraints (operational: filtering algorithms)
- Search
- Set variables
- Symmetry breaking

Outline

1. Symmetries in CSPs

2. Group theory

3. Avoiding symmetries
   ...by Reformulation
   ...by static Symmetry Breaking
   ...during Search (SBDS)
   ...by Dominance Detection (SBDD)

Symmetries

Example

$$\mathcal{P} = \langle x_i \in \{1 \ldots 3\}, \forall i = 1, \ldots 3; \mathcal{C} \equiv \{x_1 = x_2 + x_3\}\rangle$$

Solutions: $(2, 1, 1)$, $(3, 1, 2)$, $(3, 2, 1)$.

Because of the symmetric nature of the plus operator, swapping the values of x_2 and x_3 gives raise to *equivalent* solutions.

- Many constraint satisfaction problem models have symmetries (some examples in a few slides)

Why removing symmetries?

Removing symmetries in constraint programming models:
inducing a preference on a (possibly singleton) subset of each solution equivalence class

Why is it useful?

- saves the solver some work by only enumerating non-symmetric solutions, and then generating the symmetric variants ourselves.
- reduces search by avoiding to explore equivalent states (that is, avoid the solver to also explore symmetric variants of non-solution states!)

Symmetry Example: Social Golfer Problem

Problem statement

Given:

- g groups of
- s golf players,
- and w weeks.

All players play once a week and we do not want that two players play in the same group more than once.

Consider the model with a three-dimensional matrix of integer variables X_{ijk} $i \in \{1, \ldots, w\}, j \in \{1, \ldots, g\}, k \in \{1, \ldots s\}$ of integer variables $\{1, \ldots g \times s\}$ representing the player playing as k-th player during week i in group j.

Symmetry Example: Social Golfer Problem

- $g = 5$
- $s = 3$
- $\rightsquigarrow$ players $0..14$
- $w = 7$

	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
week 2	0	3	6	1	4	9	2	7	12	5	10	13	8	11	14
week 3	0	4	13	1	3	11	2	6	10	5	8	12	7	9	14
week 4	0	5	14	1	10	12	2	3	8	4	7	11	6	9	13
week 5	0	7	10	1	8	13	2	4	14	3	9	12	5	6	11
week 6	0	8	9	1	5	7	2	11	13	3	10	14	4	6	12
week 7	0	11	12	1	6	14	2	5	9	3	7	13	4	8	10

Symmetry Example: Social Golfer Problem

Permuting position in group

	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
week 2	0	3	6	1	4	9	2	7	12	5	10	13	8	11	14
week 3	0	4	13	1	3	11	2	6	10	5	8	12	7	9	14
week 4	0	5	14	1	10	12	2	3	8	4	7	11	6	9	13
week 5	0	7	10	1	8	13	2	4	14	3	9	12	5	6	11
week 6	0	8	9	1	5	7	2	11	13	3	10	14	4	6	12
week 7	0	11	12	1	6	14	2	5	9	3	7	13	4	8	10

Symmetry Example: Social Golfer Problem

Permuting position in group

	group 1			group 2			group 3			group 4			group 5		
week 1	2	1	0	3	4	5	6	7	8	9	10	11	12	13	14
week 2	6	3	0	1	4	9	2	7	12	5	10	13	8	11	14
week 3	13	4	0	1	3	11	2	6	10	5	8	12	7	9	14
week 4	14	5	0	1	10	12	2	3	8	4	7	11	6	9	13
week 5	10	7	0	1	8	13	2	4	14	3	9	12	5	6	11
week 6	9	8	0	1	5	7	2	11	13	3	10	14	4	6	12
week 7	12	11	0	1	6	14	2	5	9	3	7	13	4	8	10

Symmetry Example: Social Golfer Problem

Permuting groups

	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
week 2	0	3	6	1	4	9	2	7	12	5	10	13	8	11	14
week 3	0	4	13	1	3	11	2	6	10	5	8	12	7	9	14
week 4	0	5	14	1	10	12	2	3	8	4	7	11	6	9	13
week 5	0	7	10	1	8	13	2	4	14	3	9	12	5	6	11
week 6	0	8	9	1	5	7	2	11	13	3	10	14	4	6	12
week 7	0	11	12	1	6	14	2	5	9	3	7	13	4	8	10

Symmetry Example: Social Golfer Problem

Permuting groups

	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	9	10	11	6	7	8	3	4	5	12	13	14
week 2	0	3	6	5	10	13	2	7	12	1	4	9	8	11	14
week 3	0	4	13	5	8	12	2	6	10	1	3	11	7	9	14
week 4	0	5	14	4	7	11	2	3	8	1	10	12	6	9	13
week 5	0	7	10	3	9	12	2	4	14	1	8	13	5	6	11
week 6	0	8	9	3	10	14	2	11	13	1	5	7	4	6	12
week 7	0	11	12	3	7	13	2	5	9	1	6	14	4	8	10

Symmetry Example: Social Golfer Problem

Permuting weeks

	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
week 2	0	3	6	1	4	9	2	7	12	5	10	13	8	11	14
week 3	0	4	13	1	3	11	2	6	10	5	8	12	7	9	14
week 4	0	5	14	1	10	12	2	3	8	4	7	11	6	9	13
week 5	0	7	10	1	8	13	2	4	14	3	9	12	5	6	11
week 6	0	8	9	1	5	7	2	11	13	3	10	14	4	6	12
week 7	0	11	12	1	6	14	2	5	9	3	7	13	4	8	10

Symmetry Example: Social Golfer Problem

Permuting weeks

	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
week 2	0	7	10	1	8	13	2	4	14	3	9	12	5	6	11
week 3	0	4	13	1	3	11	2	6	10	5	8	12	7	9	14
week 4	0	5	14	1	10	12	2	3	8	4	7	11	6	9	13
week 5	0	3	6	1	4	9	2	7	12	5	10	13	8	11	14
week 6	0	8	9	1	5	7	2	11	13	3	10	14	4	6	12
week 7	0	11	12	1	6	14	2	5	9	3	7	13	4	8	10

Symmetry Example: Social Golfer Problem

Permuting players

	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
week 2	0	3	6	1	4	9	2	7	12	5	10	13	8	11	14
week 3	0	4	13	1	3	11	2	6	10	5	8	12	7	9	14
week 4	0	5	14	1	10	12	2	3	8	4	7	11	6	9	13
week 5	0	7	10	1	8	13	2	4	14	3	9	12	5	6	11
week 6	0	8	9	1	5	7	2	11	13	3	10	14	4	6	12
week 7	0	11	12	1	6	14	2	5	9	3	7	13	4	8	10

Symmetry Example: Social Golfer Problem

Permuting players

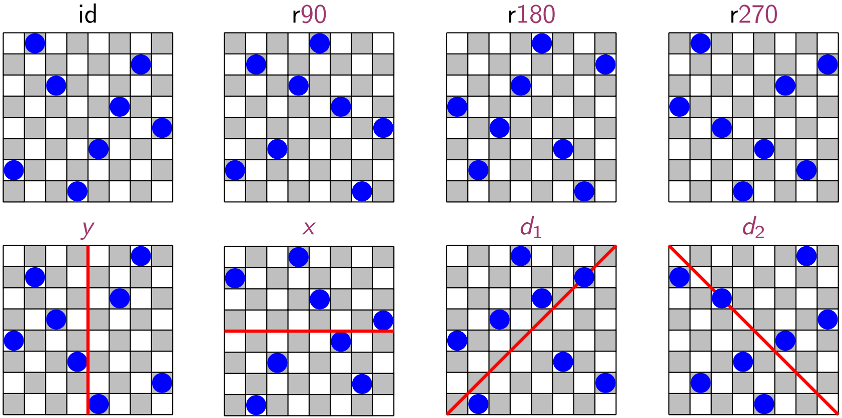
	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	3	4	5	6	9	8	7	10	11	12	13	14
week 2	0	3	6	1	4	7	2	9	12	5	10	13	8	11	14
week 3	0	4	13	1	3	11	2	6	10	5	8	12	9	7	14
week 4	0	5	14	1	10	12	2	3	8	4	9	11	6	7	13
week 5	0	9	10	1	8	13	2	4	14	3	7	12	5	6	11
week 6	0	8	7	1	5	9	2	11	13	3	10	14	4	6	12
week 7	0	11	12	1	6	14	2	5	7	3	9	13	4	8	10

Symmetry Example: Social Golfer Problem

Number of (equivalent) solutions:

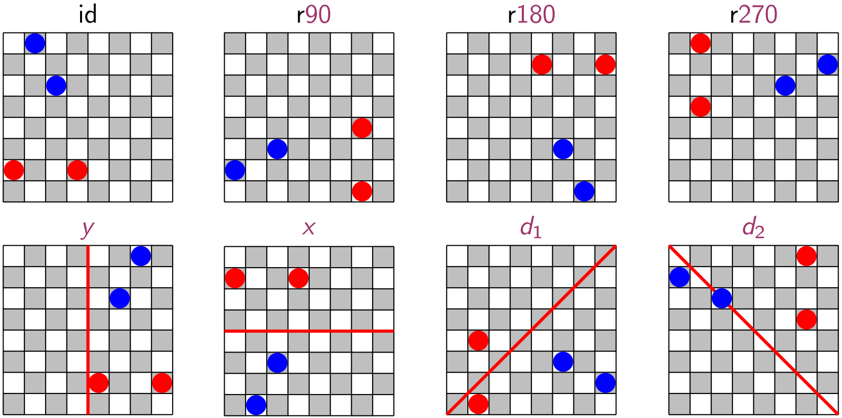
- Permuting positions: $3! \cdot 5 = 30$
- Permuting groups: $5! = 120$
- Permuting weeks: $7! = 5040$
- Permuting players: $15! = 1,307,674,368,000$

Symmetry Example: n-Queens



Symmetry Example: n-Queens

Symmetric failure



Symmetries: general considerations

- Widespread
  - Inherent in the problem (n-Queens, chessboard)
  - Artifact of the model (Social Golfer: order of players in groups)
- Different types:
  - variable symmetry (swapping variables)
  - value symmetry (permuting values)

Types of symmetries

- Variable symmetry: permuting variables is solution invariant

$$\{x_i = v_i\} \in sol(P) \iff \{x_{\sigma(i)} = v_i\} \in sol(P)$$

  eg: first three symmetries in golfers
- Value symmetry: permuting values is solution invariant

$$\{x_i = v_i\} \in sol(P) \iff \{x_i = \sigma(v_i)\} \in sol(P)$$

  eg: graph coloring, player symmetry in golfers
- Variable/value symmetry: both variables and values permutation is solution invariant

$$\{x_i = v_i\} \in sol(P) \iff \{x_{\sigma_1(i)} = \sigma_2(v_i)\} \in sol(P)$$

  eg: n-queens

Outline

1. Symmetries in CSPs

2. Group theory

3. Avoiding symmetries
   ...by Reformulation
   ...by static Symmetry Breaking
   ...during Search (SBDS)
   ...by Dominance Detection (SBDD)

Group basics

Group

A set G and an associated operation $\otimes$ form a group if

- G is closed under $\otimes$, i.e., $a, b \in G \Rightarrow a \otimes b \in G$
- $\otimes$ is associative, i.e., $a, b, c \in G \Rightarrow (a \otimes b) \otimes c = a \otimes (b \otimes c)$
- G has an identity ι_G, such that $a \in G \Rightarrow a \otimes \iota_G = \iota_G \otimes a = a$
- every element has an inverse, i.e., $a \in G \Rightarrow \exists a^{-1} : a \otimes a^{-1} = a^{-1} \otimes a = \iota_G$

Permutations

Permutation representations:

Cauchy's two-line notation:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 4 & 1 & 8 & 5 & 2 & 9 & 6 & 3 \end{pmatrix}$$

element 1 maps to 7, 7 to 9, 9 to 3, 3 to 1.

Cycle notation:

$$(2, 4, 8, 6)(1, 7, 9, 3)(5)$$

set of cycles derived from the two-line notation indicating the mapping, ie, 2 becomes 4, 4 becomes 8, etc.

The set of all permutations of a finite set S of objects together with composition form a group.

Group properties for permutations with composition $\circ$ as operation. Let f and g be two permutations, p a point:

- $f \circ g$ composition
- $p^{f \circ g} = (p^f)^g$
- $id = \iota$
- $f \circ f^{-1} = id$ inverse (in Cauchy form, swap the two rows and reorder the first; in cycle notation, reverse the order of each cycle.)
- associativity: $f \circ (g \circ h) = (f \circ g) \circ h$

- $|G|$ is the order of a group, ie, number of elements in the set G
- Set S_n of all permutations of n objects is called a symmetry group over n elements. $|S_n| = n!$
- Any subgroup of a permutation group defines a permutation group
- The set of symmetries in n-queens defines a permutation group: $\{id, r90, r180, r270, x, y, d_1, d_2\}$
- symmetries define a permutation of a set of points.
- p a point in the solution space, $g \in G$ a permutation, p^g the point to which p is moved under g. Eg: $\{1,3,8\}^{r90} = \{1^{r90}, 3^{r90}, 8^{r90}\} = \{7,1,6\}$

Generators

Generators

A set $S \subseteq G$ is called a generator of group G iff

$$\forall g \in G \quad \exists S' \subseteq S : \quad g = \bigotimes_{s \in S'} s$$

Generators describe groups in a compact form.
For example:

- Generator of chessboard symmetries: $\{r90, d1\}$
- $G =< s >$
- There is always a generator of $\log_2(|G|)$ size or smaller.

Orbits

Orbits

The orbit of an element with respect to a permutation group G is

$$O^G = \{p^g \mid g \in G\}$$

The orbit of a set of elements (called also points) is defined accordingly.

Orbits are the set of elements encountered by starting from one element and moving through different permutations.

Outline

How to avoid symmetry

Never explore a state that is the symmetric of one already explored

- Model reformulation
- Addition of constraints (static symmetry breaking)
- During search (dynamic symmetry breaking)
- By dominance detection (dynamic symmetry breaking)

Outline

1. Symmetries in CSPs
2. Group theory
3. Avoiding symmetries
    ...by Reformulation
    ...by static Symmetry Breaking
    ...during Search (SBDS)
    ...by Dominance Detection (SBDD)

Model reformulation

- Use set variables (inherently unordered)
  - In the Social Golfers example: groups can be represented as sets
  - Only within group symmetry has been removed, but not the groups/weeks/player ones
- Solve a different problem (try to redefine the problem avoiding symmetries)
- Solve the dual problem

Solve a different problem: example

A series is a sequence of twelve tone names (pitch classes) of the chromatic scale, in which each pitch class occurs exactly once. In an all-interval series, also all eleven intervals between the twelve pitches are pairwise distinct.

All-different series

In general words, we are asked to find a permutation of the integers $\{0, \ldots, n\}$, such that the differences between adjacent numbers are a permutation of $\{1, \ldots, n\}$.

0 10 1 9 2 8 3 7 4 6 5

10 9 8 7 6 5 4 3 2 1

The problem has many symmetric solutions, e.g. reverse values, "invert" from 10, shifting (according to a pivot), ...

0 10 1 9 2 8 3 7 4 6 5   3 7 4 6 5 0 10 1 9 2 8

10 9 8 7 6 5 4 3 2 1   4 3 2 1 5 10 9 8 7 6

Solve a different problem: example

All-different series: new formulation

Find a permutation of the integers $\{0, \ldots, n\}$ such that:

- the permutation starts with 0, n, 1
- the differences $|x_{i+1} - x_i|$ and $|x_n - x_0|$ are in $\{1, \ldots, n\}$
- exactly one difference occurs twice

This extracts solutions from the original problem with a specific structure

Solve dual problem

- Mainly for value symmetries
- Example: players in golfers
- Consider the dual problem w.r.t. each value v
  - Introduce a set X_v such that
    $$i \in X_v \iff y_i = v$$
    (y_i are the original variables)
- Applicable when constraints can be stated easily on the dual problem

Outline

Symmetry breaking constraints

- Rule out symmetric solutions by adding further constraints to the original model.
- Assumption: domains are ordered

Lexcographic ordering between vectors: $\vec{x} \preceq_{lex} \vec{y}$

$$
\begin{array}{ll}
\vec{x} \prec_{lex} \vec{y} & \text{iff } \exists \ell \geq 1 \text{ such that for all } i < \ell, x_i = y_i \text{ and } x_\ell < y_\ell \\
\vec{x} =_{lex} \vec{y} & \text{iff for all } i = 1, \ldots, n \;\; x_i = y_i
\end{array}
$$

Lex-leader constraints

Let Σ be the set of all variable symmetry permutations
These symmetries are broken by imposing:

$$[x_1, \ldots, x_n] \preceq_{lex} [x_{\sigma(1)}, \ldots x_{\sigma(n)}], \quad \forall \sigma \in \Sigma$$

Only the lexicographically smallest solution, called lex-leader is preserved

Example:
Consider a CSP with a matrix of variables (a simplified case of social golphers):

$$\begin{matrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{matrix} \qquad \begin{matrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{matrix} \qquad \begin{matrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{matrix}$$

Assume the rows are interchangeable, ie, $[x_1, x_2, x_3] \leftrightarrow [x_4, x_5, x_6] \leftrightarrow [x_7, x_8, x_9]$

Symmetry breaking constraints:

$$[x_1, x_2, x_3] \preceq_{lex} [x_4, x_5, x_6]$$
$$[x_1, x_2, x_3] \preceq_{lex} [x_7, x_8, x_9]$$
$$[x_4, x_5, x_6] \preceq_{lex} [x_7, x_8, x_9]$$

Symmetry with all different

- Distinct integers, $\sigma(1) \neq 1$: $[x_1, \ldots, x_n] \preceq_{lex} [x_{\sigma(1)}, \ldots x_{\sigma(n)}] \iff x_1 < x_{\sigma(1)}$
- Disjoint integer sets, $\sigma(1) \neq 1$: $[x_1, \ldots, x_n] \preceq_{lex} [x_{\sigma(1)}, \ldots x_{\sigma(n)}] \iff \min(x_1) < \min(x_{\sigma(1)})$
- Arbitrary integers or sets: special propagators

Lex-leader constraints: examples

- n-Queens: $\sigma(i) = n - i + 1$ (eliminate symmetry rotation on y)

$$[q_1, \dots q_n] \preceq_{lex} [q_{\sigma(1)}, \dots q_{\sigma(n)}] = [q_n, \dots, q_1]$$

$$\implies q_1 < q_n$$

- All-Intervals:

$$|x_2 - x_1| > |x_n - x_{n-1}|$$

- k-coloring problem: dual representation in color classes $C_1, C_2, \dots, C_k$

$$[C_1, C_2, \dots, C_k] \preceq_{lex} [C_{\sigma(1)}, C_{\sigma(2)}, \dots, C_{\sigma(k)}] \implies \min(C_1) < \min(C_{\sigma(1)})$$

where the implication is due to the fact that the color classes are disjoint

In MiniZinc

n-queens problem with a different solution representation

```
int: n;
set of int: N = 1..n;
array[N,N] of var bool: t;
sum(i,j in N)(t[i,j]) = n;
.
.
.
solve satisfy;
```

```
% no two traps on the same row
forall(i in N)(sum(j in N)(t[i,j]) <= 1);
% no two traps on the same column
forall(j in N)(sum(i in N)(t[i,j]) <= 1);
% no two traps on same diagonal
forall(k in 1-n..n-1)
(sum(i,j in N where i-j=k)(t[i,j]) <= 1);
forall(k in 2..2*n)
(sum(i,j in N where i+j=k)(t[i,j]) <= 1);
```

For r90:

$$
\begin{matrix}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} & x_{1,7} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} & x_{2,7} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} & x_{3,7} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} & x_{4,6} & x_{4,7} \\
x_{5,1} & x_{5,2} & x_{5,3} & x_{5,4} & x_{5,5} & x_{5,6} & x_{5,7} \\
x_{6,1} & x_{6,2} & x_{6,3} & x_{6,4} & x_{6,5} & x_{6,6} & x_{6,7} \\
x_{7,1} & x_{7,2} & x_{7,3} & x_{7,4} & x_{7,5} & x_{7,6} & x_{7,7}
\end{matrix}
\quad \leq_{\mathtt{lex}} \quad
\begin{matrix}
x_{7,1} & x_{6,1} & x_{5,1} & x_{4,1} & x_{3,1} & x_{2,1} & x_{1,1} \\
x_{7,2} & x_{6,2} & x_{5,2} & x_{4,2} & x_{3,2} & x_{2,2} & x_{1,2} \\
x_{7,3} & x_{6,3} & x_{5,3} & x_{4,3} & x_{3,3} & x_{2,3} & x_{1,3} \\
x_{7,4} & x_{6,4} & x_{5,4} & x_{4,4} & x_{3,4} & x_{2,4} & x_{1,4} \\
x_{7,5} & x_{6,5} & x_{5,5} & x_{4,5} & x_{3,5} & x_{2,5} & x_{1,5} \\
x_{7,6} & x_{6,6} & x_{5,6} & x_{4,6} & x_{3,6} & x_{2,6} & x_{1,6} \\
x_{7,7} & x_{6,7} & x_{5,7} & x_{4,7} & x_{3,7} & x_{2,7} & x_{1,7}
\end{matrix}
$$

In MiniZinc

```
% solution <=lex r90 version
let { array[N,N] of var bool: s; } in
forall(i,j in N)(s[i,j] = t[j,n+1-i]) /\
lex_lesseq(array1d(t), array1d(s));
```

In Gecode

- Lexicographic constraints between variable arrays. (where the sizes of x and y can be different), If x and y are integer variable arrays

```
rel(home, x, IRT_LE, y);
```

Social Golfers

In Gecode

- Using set variables to model the groups avoids introducing symmetry among the players in a group.

```
SetVarArray groups(home,g*w,IntSet::empty,0,g*s-1,s,s);
Matrix<SetVarArray> schedule(groups,g,w);
```

- Within a week, the order of the groups is irrelevant. Static order requiring all minimal elements of each group are ordered increasingly $\min(schedule(g, w)) < \min(schedule(g + 1, w))$

```
for (int j=0; j<w; j++) {
  IntVarArgs m(g);
  for (int i=0; i<g; i++)
    m[i] = expr(home, min(schedule(i,j)));
  rel(home, m, IRT_LE);
}
```

- similarly, the order of the weeks is irrelevant, hence order on group elements (remove $\{0\}$ as from above it will be always in $schedule(0, j)$

```
IntVarArgs m(w);
for (int j=0; j<w; j++)
  m[j] = expr(home, min(schedule(0,j)-IntSet(0,0)));
rel(home, m, IRT_LE);
```

Value symmetries

- Same idea:

$$[x_1, \ldots, x_n] \preceq_{lex} [\sigma(x_1), \ldots \sigma(x_n)], \quad \forall \sigma \in \Sigma$$

- how to implement $\sigma(x_i)$?
- Value symmetries become variable symmetries in inverse formulation of permutation problems or other view points
- otherwise, see next slides for permutation symmetries
- or element constraint to implement $\sigma(x_i)$

Example

$\sigma(v) = n - v$

3 7 4 6 5 0 10 1 9 2 8   7 3 6 4 5 10 0 9 1 8 2

4 3 2 1 5 10 9 8 7 6   4 3 2 1 5 10 9 8 7 6

$\sigma = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$

$[x_0, \ldots, x_{n-1}] \preceq_{lex} [\sigma(x_0), \ldots \sigma(x_{n-1})] \iff x_0 < \sigma(x_0) \iff x_0 < \sigma[x_0]$

In Gecode

- x is an array of set variables and c is an array of integers

```
precede(home, x, c);
```

  it is enforced that c_k precedes c_{k+1} in x for $0 \leq k < |c| - 1$

Value Symmetries in MiniZinc

Example: graph coloring, direct representation

```
include "value_precede_chain.mzn";
include "globals.mzn";

enum REGION = { WA, NT, SA, Q, NSW, V, T };
set of int: COLOR = 1..3; %card(REGION);

% Neighboring regions
array [int, int] of REGION: neighbors = [| WA, NT | WA, SA | NT, SA | NT, Q | SA, Q | SA, NSW | SA,
    V | Q, NSW | NSW, V|];

% Color of each Region
array [REGION] of var COLOR: color;
% Number of colors used
%var 1..card(COLOR): n_colors; % :: output = card({ c | c in color });

%constraint nvalue(color) = 3; %
% Neighboring regions have different colours
constraint forall (i in index_set_1of2(neighbors))
                  (color[neighbors[i, 1]] != color[neighbors[i, 2]]);


constraint symmetry_breaking_constraint(
        value_precede_chain(COLOR, color)
        );
%bool: mzn_ignore_symmetry_breaking_constraints=false;
% Use as few colors as possible
solve satisfy;
```

Run with:

```
> minizinc --solver org.gecode.gecode --all-solutions coloring.mzn -s -D "
    mzn_ignore_symmetry_breaking_constraints=true;" -o coloring.ozn
```

Output (18 solutions):

```
color = [WA: 3, NT: 2, SA: 1, Q: 3, NSW: 2, V: 3, T: 1];
color = [WA: 3, NT: 2, SA: 1, Q: 3, NSW: 2, V: 3, T: 2];
color = [WA: 3, NT: 2, SA: 1, Q: 3, NSW: 2, V: 3, T: 3];
color = [WA: 2, NT: 3, SA: 1, Q: 2, NSW: 3, V: 2, T: 1];
color = [WA: 2, NT: 3, SA: 1, Q: 2, NSW: 3, V: 2, T: 2];
color = [WA: 2, NT: 3, SA: 1, Q: 2, NSW: 3, V: 2, T: 3];
color = [WA: 3, NT: 1, SA: 2, Q: 3, NSW: 1, V: 3, T: 1];
color = [WA: 3, NT: 1, SA: 2, Q: 3, NSW: 1, V: 3, T: 2];
color = [WA: 3, NT: 1, SA: 2, Q: 3, NSW: 1, V: 3, T: 3];
color = [WA: 1, NT: 3, SA: 2, Q: 1, NSW: 3, V: 1, T: 1];
color = [WA: 1, NT: 3, SA: 2, Q: 1, NSW: 3, V: 1, T: 2];
color = [WA: 1, NT: 3, SA: 2, Q: 1, NSW: 3, V: 1, T: 3];
color = [WA: 2, NT: 1, SA: 3, Q: 2, NSW: 1, V: 2, T: 1];
color = [WA: 2, NT: 1, SA: 3, Q: 2, NSW: 1, V: 2, T: 2];
color = [WA: 2, NT: 1, SA: 3, Q: 2, NSW: 1, V: 2, T: 3];
color = [WA: 1, NT: 2, SA: 3, Q: 1, NSW: 2, V: 1, T: 1];
color = [WA: 1, NT: 2, SA: 3, Q: 1, NSW: 2, V: 1, T: 2];
color = [WA: 1, NT: 2, SA: 3, Q: 1, NSW: 2, V: 1, T: 3];
```

```
> minizinc --solver org.gecode.gecode --all-solutions coloring.mzn -s -D "
    mzn_ignore_symmetry_breaking_constraints=false;" -o coloring.ozn
```

Output (3 solutions):

```
color = [WA: 1, NT: 2, SA: 3, Q: 1, NSW: 2, V: 1, T: 1];
color = [WA: 1, NT: 2, SA: 3, Q: 1, NSW: 2, V: 1, T: 2];
color = [WA: 1, NT: 2, SA: 3, Q: 1, NSW: 2, V: 1, T: 3];
```

Social Golfers
In Gecode

- the players can be permuted arbitrarily.

```
precede(home, groups, IntArgs::create(g*s, 0));
```

$c = (0, \ldots, 14)$: It enforces that for any pair of players c_k and c_{k+1}, $0 \leq k \leq 14$, c_{k+1} can appear in a group without c_k only if there is an earlier group where c_k appears without c_{k+1}.
Eg, 9 appears in a group without 7 but 7 should appear earlier, hence the constraint is not satisfied.
In Minizinc:

```
predicate value_precede_chain(array [int] of $$E: c, array [int] of var set of $$E: x)
```

	group 1			group 2			group 3			group 4			group 5		
week 1	0	1	2	3	4	5	6	9	8	7	10	11	12	13	14
week 2	0	3	6	1	4	7	2	9	12	5	10	13	8	11	14
week 3	0	4	13	1	3	11	2	6	10	5	8	12	9	7	14
week 4	0	5	14	1	10	12	2	3	8	4	9	11	6	7	13
week 5	0	9	10	1	8	13	2	4	14	3	7	12	5	6	11
week 6	0	8	7	1	5	9	2	11	13	3	10	14	4	6	12
week 7	0	11	12	1	6	14	2	5	7	3	9	13	4	8	10

Pros and Cons

- Good: for each symmetry, only one solution remains
- Bad:
  may have to add many constraints
  remaining solution may not be the first one according to branching heuristic!
- Especially bad with dynamic variable selection (like first-fail heuristics)

For Minizinc, see:

- `https://www.minizinc.org/doc-latest/en/efficient.html#symmetry`
- `https://www.minizinc.org/doc-latest/en/lib-globals-lexicographic.html`

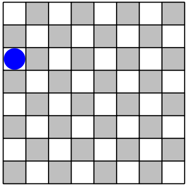
Outline

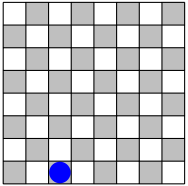
Symmetry Breaking During Search (SBDS)

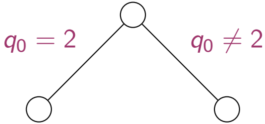
- Add constraints during backtracking to prevent the visit of symmetric search states
- Similar idea to branch-and-bound
- Pros: Works with every type of symmetry
- Cons: Can result in a huge number of constraints to be added, and all symmetries have to be specified explicitly

SBDS Example: *n*-Queens

Goal: Eliminate r90: $\{q_i = j\} \in sol(n\text{-Queens}) \iff \{q_j = n - i\} \in sol(n\text{-Queens})$

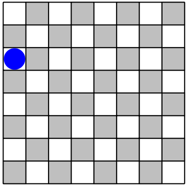


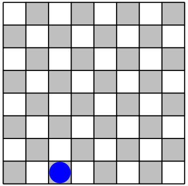


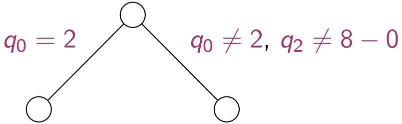


SBDS Example: *n*-Queens

Goal: Eliminate r90: $\{q_i = j\} \in sol(n\text{-Queens}) \iff \{q_j = n - i\} \in sol(n\text{-Queens})$

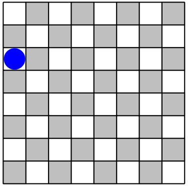


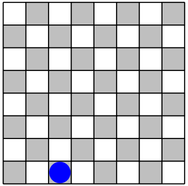


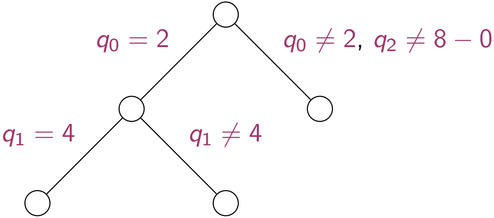


SBDS Example: n-Queens

Goal: Eliminate r90: $\{q_i = j\} \in sol(n\text{-Queens}) \iff \{q_j = n - i\} \in sol(n\text{-Queens})$

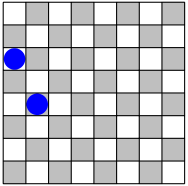


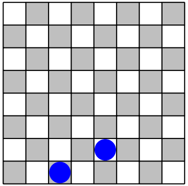


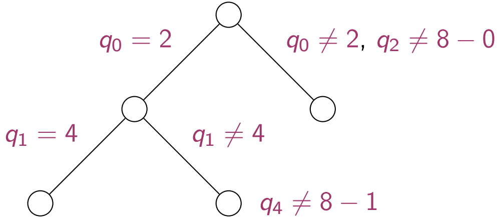


SBDS Example: n-Queens

Goal: Eliminate r90: $\{q_i = j\} \in sol(n\text{-Queens}) \iff \{q_j = n - i\} \in sol(n\text{-Queens})$

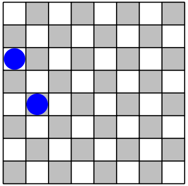


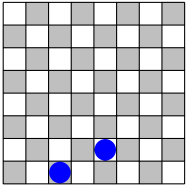


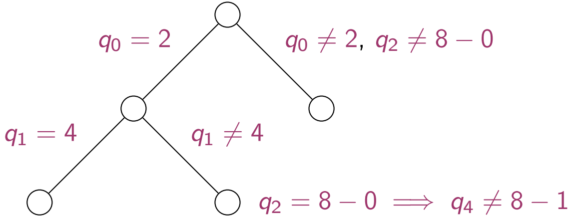


SBDS Example: *n*-Queens

Goal: Eliminate r90: $\{q_i = j\} \in sol(n\text{-Queens}) \iff \{q_j = n - i\} \in sol(n\text{-Queens})$







Too strict: we need to post the whole path:
$\neg(q_0 = 2 \wedge q_1 = 4) \rightsquigarrow (q_0 = 2 \implies q_1 \neq 4)^{r90}$

SBDS in group theory perspective

$$(A \implies var \neq val) \implies (A \implies var \neq val)^g$$

We do not need to add the full form. We operate dynamically:
At the choice point c backtracking from $var = val$ we know that A is true and $var \neq val$ is also true, hence we add:

$$A^g \implies (var \neq val)^g$$

SBDS

For each symmetry g, and a current partial assignment A and choice c, post the constraint:

$$g(A) \implies \neg g(c)$$

Only interested in different $g(A)$ and $g(c)$

- compute the orbit of the current partial assignment A

Lightweight Dynamic Symmetry Breaking

In Gecode

Dynamic symmetry breaking: given a specification of the symmetries, avoid visiting symmetric states during the search

- break value symmetry (that is, values that are interchangeable)

```
Symmetries syms;
syms << ValueSymmetry(IntArgs::create(n,0));
branch(* this, x, INT_VAR_NONE(), INT_VAL_MIN(), syms);
```

- break variable symmetry (that is, certain sequences of variables are interchangeable):

```
IntVarArgs rows;
for (int r = 0; r < m.height(); r++)
  rows << m.row(r);
syms << VariableSequenceSymmetry(rows, m.width());
IntVarArgs cols;
for (int c = 0; c < m.width(); c++)
  cols << m.col(c);
syms << VariableSequenceSymmetry(cols, m.height());
```

- See sec. 8.10.1 for other possibilities
- Combining LDSB with other forms of symmetry breaking — such as static ordering constraints — can cause the search to miss some sol.

Outline

1. Symmetries in CSPs
2. Group theory
3. Avoiding symmetries
    ...by Reformulation
    ...by static Symmetry Breaking
    ...during Search (SBDS)
    ...by Dominance Detection (SBDD)

Symmetry Breaking by Dominance Detection (SBDD)

- Do not explore subtrees dominated by a previously visited node
- Multiple definitions of *dominance* are possible
- Pros: No constraints added, very configurable
- Cons: Storage of previous states, checking dominance can be expensive

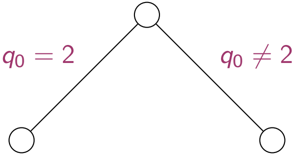
The idea is similar to *no goods*.
It can be used for propagation.

Ingredients

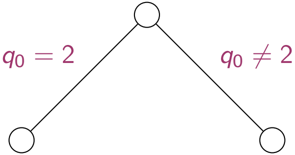
Idea: Perform a check at every node in the search tree to see if the node about to be explored is symmetrically equivalent to one already explored. If so prune this branch. Need only to store nodes at the root of fully explored subtrees.

- No-good: A node v is a no-good w.r.t. a node n if there exists an ancestor n_a of n s.t. v is the left hand child of n_a and v is not an ancestor of n.
- $\delta(v)$ set of decisions labelling the path from the root of the tree to the node v
- $\Delta(v)$ set of variables whose domain is reduced to a singleton at node v.
- Dominance:
  a node n is dominated if there exists a no-good v w.r.t. n and a symmetry g s.t. $(\delta(v))^g \subseteq \Delta(n)$
- Database T of already seen domains

SBDD Example: n-Queens

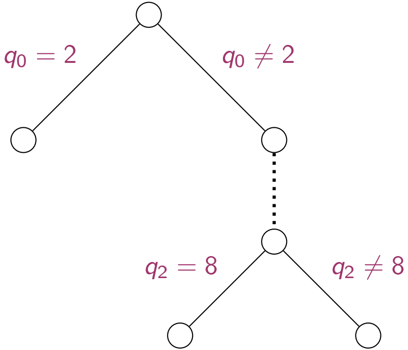


SBDD Example: n-Queens



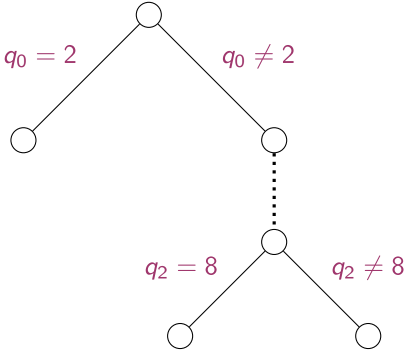
$$T = \{\{q_0 = 2\}\}$$

SBDD Example: n-Queens



$T = \{\{q_0 = 2\}\}$

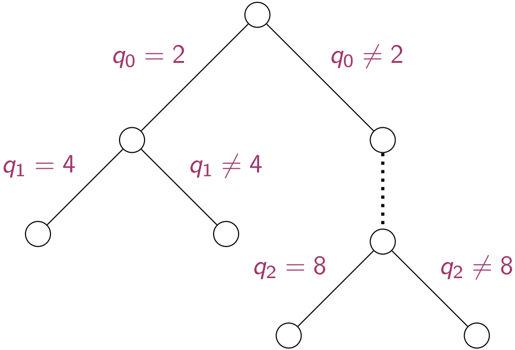
SBDD Example: n-Queens



$T = \{\{q_0 = 2\}\}$

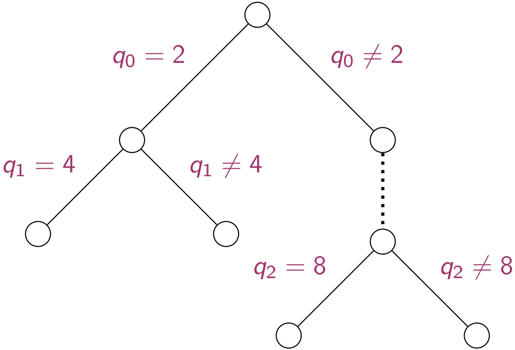
Dominated

SBDD Example: n-Queens



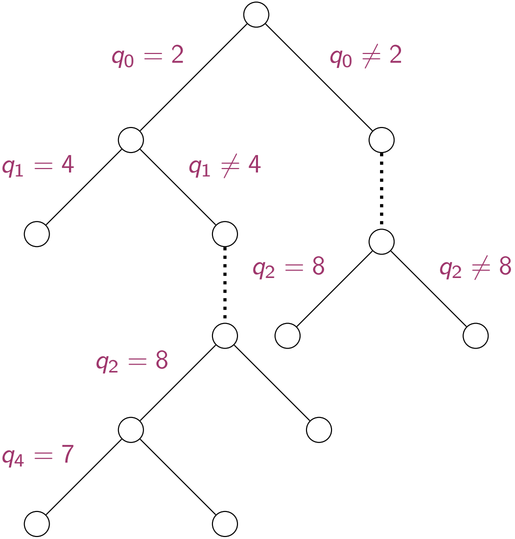
$T = \{\{q_0 = 2, q_1 = 4\}\}$

SBDD Example: n-Queens



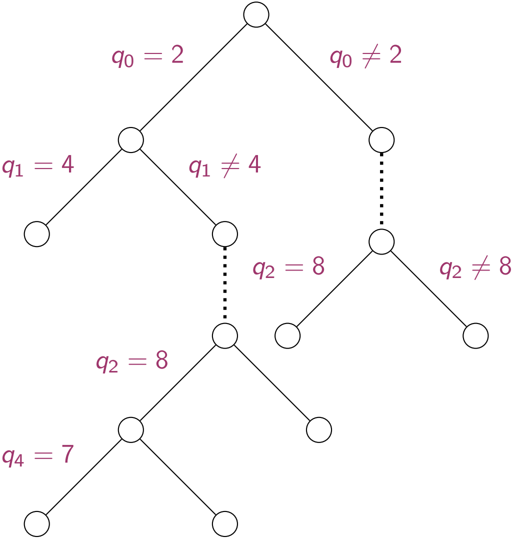
$T = \{\{q_0 = 2, q_1 = 4\}\}$

SBDD Example: n-Queens



$T = \{\{q_0 = 2, q_1 = 4\}\}$

SBDD Example: n-Queens



$T = \{\{q_0 = 2, q_1 = 4\}\}$

Dominated

SBDD in the group theory perspective

SBDD

A domain d dominates the current node c if c is in the orbit of d

Detection:
function $\Phi : \mathrm{Dom} \times \mathrm{Dom} \mapsto \mathbb{B}$
such that $\Phi(\delta(v), \Delta(n)) = \mathit{true}$ iff $\delta(v)$ dominates $\Delta(n)$ under some symmetry σ.

Optimization: only keep domains left-adjacent to the path from the root to the current node

Pros and Cons

- Good: No constraints added
- Good: Handles all kinds of symmetry
- Good: Very configurable (by implementing)
- Bad: Still all symmetries must be encoded
- Bad: Checking dominance at each node may be expensive

Other Dominance Constraints

Example: Maximizing cooperation in Road guarding

```
forall (i in 2..n-2)(
  forall (j in i+1..n-1)(
   coop[road[i-1], road[i]] +
   coop[road[j], road[j+1]]
   >=
   coop[road[i-1], road[j]] +
   coop[road[i], road[j+1]]
   )
 );
```

References

Backofen W. (2002). **Excluding symmetries in constraint-based search**. *Constraints*, (3).

Barnier N. and Brisset P. (2005). **Solving kirkman's schoolgirl problem in a few seconds**. *Constraints*, (10), pp. 7–21.

Gent I.P., Petrie K.E., and Puget J.F. (2006). **Symmetry in constraint programming**. In *Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh, chap. 10, pp. 329–376. Elsevier.