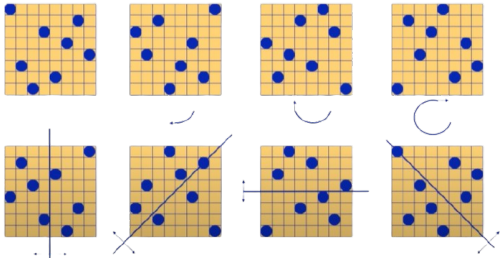
L06: Symmetry and dominance breaking



Prof. Tias Guns and Dr. Dimos Tsouros

KU LEUVEN

Partly based on slides from Pierre Flener, Uppsala University.

Outline

1. Introduction

2. Symmetries

3. Symmetry Breaking
   - Symmetry Breaking by Reformulation
   - Symmetry Breaking by Constraints

4. Dominance Breaking

5. Recap

Quick Recap

1. Modelling: express problem in terms of
   - parameters,
   - decision variables, with their domains
   - constraints, and
   - (optionally) objective.
2. Solving: solve using a state-of-the-art solver.

From a Problem to a Model

What is a good model for a constraint problem?

- A model that correctly represents the problem.
- A model that is easy to understand and maintain.
- A model that is solved efficiently.

Modelling

Modelling is still more a craft than a science:

- Choice of **viewpoint**: which are the decision variables and their domains?
  - Use of **auxiliary** variables ...
  - **Channel** different viewpoints ...
- Choice of the constraint formulations
  - Use **implied** constraints to reduce the search space ...
- Choice of solver and solver configuration

Make a model correct before making it efficient!

Making an efficient model

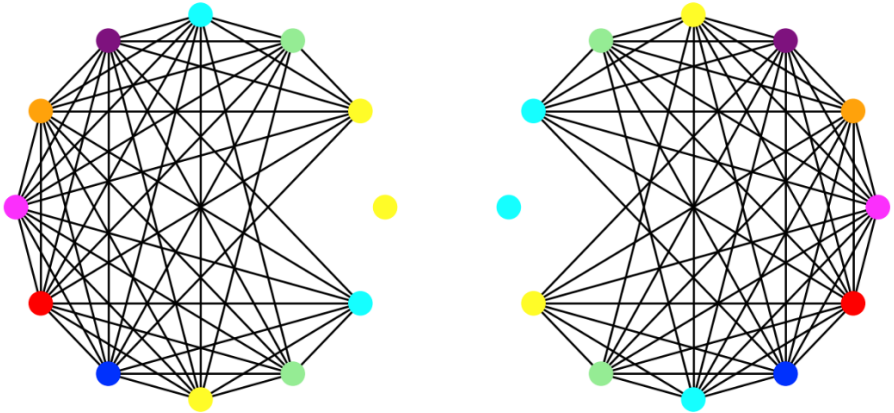
Implied constraints can reduce the search space.

Definition (recap)

Implied constraints, also called *redundant* constraints, are constraints that are entailed by the constraints defining the problem. They do ***not change the set of solutions***, and hence are logically redundant.

Is there any other way to reduce the search space, making the model more efficient?

Example: graph colouring



Exploit the problem symmetries!

Outline

1. Introduction

2. Symmetries

3. Symmetry Breaking
   - Symmetry Breaking by Reformulation
   - Symmetry Breaking by Constraints

4. Dominance Breaking

5. Recap

Symmetries

Symmetries are defined based on the concept of permutation; permuting elements of a group of variables or values can lead to symmetric solutions!

Definition

A permutation π over a discrete *set* D is a 1-1 function from D to D. It is an arrangement of the elements of the set in a specific order.

- Intuitively: just a re-arrangement of the elements. E.g.:

$$\begin{array}{rl} i: & (1 \quad 2 \quad 3 \quad 4 \quad 5) \\ \pi(i): & (4 \quad 3 \quad 2 \quad 1 \quad 5) \end{array}$$

Symmetries

Definition

A symmetry σ is a permutation of values or decision variables (or both) that preserves all constraints and solutions: it transforms (partial) solutions into (partial) solutions, and it transforms (partial) non-solutions into (partial) non-solutions.

Example

Assume a problem with 5 variables, that has 1 variable symmetry σ:

$$\begin{aligned} i &: (x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5) \\ \sigma(i) &: (x_4 \quad x_3 \quad x_2 \quad x_1 \quad x_5) \end{aligned}$$

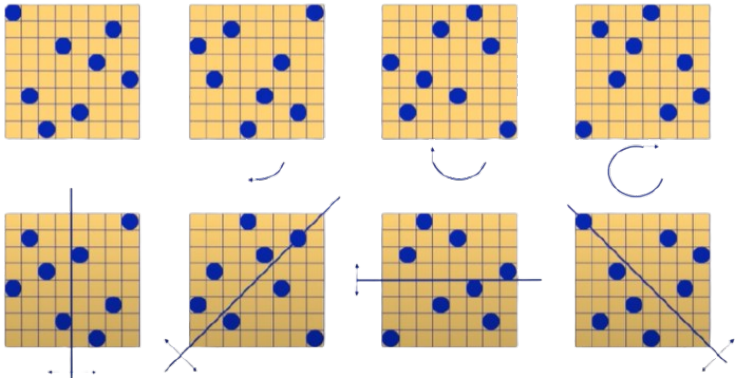
Based on the symmetry σ, each solution: $(x_1 = v_1 \quad x_2 = v_2 \quad x_3 = v_3 \quad x_4 = v_4 \quad x_5 = v_5)$

can be transformed to a symmetric solution:
$(\sigma(x_1) = v_1 \quad \sigma(x_2) = v_2 \quad \sigma(x_3) = v_3 \quad \sigma(x_4) = v_4 \quad \sigma(x_5) = v_5) =$
$(x_4 = v_1 \quad x_3 = v_2 \quad x_2 = v_3 \quad x_1 = v_4 \quad x_5 = v_5)$

Example (Variable symmetry: *n*-Queens)

We want to place *n* queens on a *nxn* chessboard such that *no queen attacks another*.

Symmetries: any reflection or rotation of an $n \times n$ board with *n* queens transforms that (non-)solution into another (non-)solution, based on the *constraints*.



Example (Continued)

The symmetries of 8-queens are the following:

id: $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \\ q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \end{pmatrix}$

r90: $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \\ q_7 & q_4 & q_1 & q_8 & q_5 & q_2 & q_9 & q_6 & q_3 \end{pmatrix}$

r180: $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \\ q_9 & q_8 & q_7 & q_6 & q_5 & q_4 & q_3 & q_2 & q_1 \end{pmatrix}$

r270: $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \\ q_3 & q_6 & q_9 & q_2 & q_5 & q_8 & q_1 & q_4 & q_7 \end{pmatrix}$

x: $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \\ q_3 & q_2 & q_1 & q_6 & q_5 & q_4 & q_9 & q_8 & q_7 \end{pmatrix}$

y: $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \\ q_7 & q_8 & q_9 & q_4 & q_5 & q_6 & q_1 & q_2 & q_3 \end{pmatrix}$

d1: $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \\ q_1 & q_4 & q_7 & q_2 & q_5 & q_8 & q_3 & q_6 & q_9 \end{pmatrix}$

d2: $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \\ q_9 & q_6 & q_3 & q_8 & q_5 & q_2 & q_7 & q_4 & q_1 \end{pmatrix}$

Value and variable symmetry

Definition

A value symmetry is a permutation of the values in a CSP that preserves all constraints and solutions. For a given value symmetry σ, $[x_i = v, \forall\ 1 \leq i < n]$ is symmetrical to $[x_i = \sigma(v), \forall\ 1 \leq i < n]$. If two values can be swapped without altering the set of solutions or violating any constraints, the CSP exhibits value symmetry.

Definition

A variable symmetry σ is a permutation of the decision variables in a CSP that preserves all constraints and solutions. For a given variable symmetry σ, $[x_i = v, \forall\ 1 \leq i < n]$ is symmetrical to $[\sigma(x_i) = v, \forall\ 1 \leq i < n]$. If two variables can be swapped without changing the set of solutions or violating any constraints, the CSP exhibits variable symmetry.

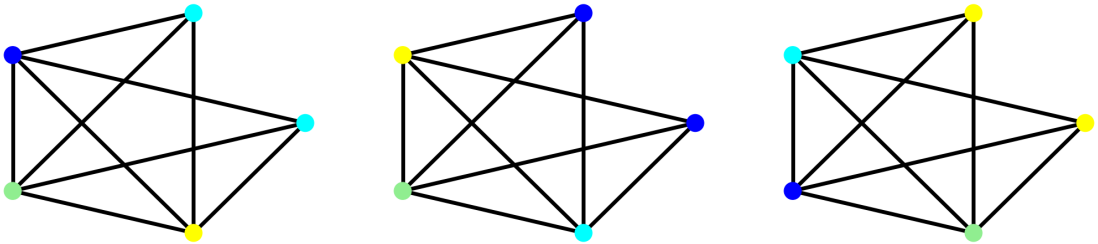
As with the variable symmetries in the example with n-queens

Example (Value symmetry: map coloring)

Use k colours to paint the countries of a map such that no neighbour countries have the same colour.

The model where the countries (as decision variables) take colours (as values) has $k!$ value symmetries because any permutation of the colours transforms a (non-)solution into another (non-)solution: the values are not distinguished.

Symmetric colouring of graphs:



Example (Variable symmetry: subset problem)

Problem definition: Find an n-element subset of a given set S, subject to some constraints.

The model encoding the subset as an array of n integer decision variables of domain S, constrained to take distinct values, has
$n!$ variable symmetries as the order of the elements does not matter in a set, but does matter in an array.

Assume $n = 3$. there are 6 variable symmetries:
Solution [1,2,3] (i.e., `x1` = 1, `x2` = 2, `x3` = 3)
is symmetric to solutions
[1,3,2], [2,1,3], [2,3,1], [3,2,1], [3,1,2]

Variable symmetriy special case: Index symmetry

Index symmetry is a special case of variable symmetry, occurring on slices of a tensor of decision variables.

Definition

Index symmetry: occurs in problems involving arrays or matrices of decision variables; permutations of **slices** of an array of decision variables preserve all constraints and solutions.

Common cases: row symmetry, column symmetry ...

Careful: Index symmetries multiply up!
If there is full row and column symmetry in an $m \times n$ array
(that is, if there are $m!$ row symmetries and $n!$ column symmetries),
then there are $m! \cdot n!$ compositions of symmetries.

Variable symmetry special case: Index symmetry

Example (Latin Squares)

Construct an $n \times n$ Latin square where each row and column contains each number from 1 to n exactly once.

The problem has $n!^2$ index symmetries because any permutation of the rows or columns ($n!$ permuations each) transforms a (non-)solution into another (non-)solution: the indices are not distinguished.

A solution

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

Swap rows 1 and 2

2	3	4	1
1	2	3	4
3	4	1	2
4	1	2	3

Swap columns 1 and 2

2	1	3	4
3	2	4	1
4	3	1	2
1	4	2	3

Symmetries

What is the origin of the symmetries?

Symmetries can be present due to different reasons

- Problem symmetries
- Instance symmetries
- Model symmetries

Problem symmetries

Definition

A problem symmetry is a symmetry of a CSP that is inherent to the problem itself and is detectable in *every model* of the problem.

The symmetries in map colouring and n-queens we discussed are problem symmetries:

- The values in graph colouring do not represent entities with different attributes; the actual colour does not matter.
- The variables in n-queens represent the queens with the same characteristics.
- Both the tasks and the machines in the scheduling problem are indistinguishable, thus symmetric.

Instance symmetries

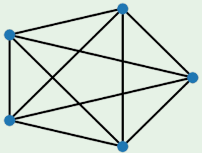
Definition

An instance symmetry is a symmetry detectable in a specific instance of a CSP. These symmetries depend on the particular values or structure of the instance rather than the general problem or its model.

Example (Graph colouring)

Problem: Use k colours to paint the nodes of a graph such that no neighbour nodes have the same colour.

Instance: Use 5 colours to paint 5 nodes that are all neighbours with each other (i.e., they form a clique).



Symmetry: Besides the value symmetry that is inherent to the graph colouring problem, we also have variable symmetry in this instance due to its structure: All variables have the same properties and thus are symmetric! Permuting the assignment of any 2 variables converts a (non-)solution to a (non-)solution.

Model symmetries

Definition

A model symmetry is a symmetry that arises from a specific representation of a CSP and is not detectable in every model of the problem.

The symmetries in the subset model are *not* problem symmetries but model symmetries: they are *not* present for every modeling choice.

Example

Subset problem: Find an n-element subset of a given set S, subject to some constraints.

The model using boolean variables B, with B_i representing if element i is part of the subset, does not have the variable symmetries of the integer model!

Model symmetries are ***introduced*** during modeling and can be avoided by using a different model (to be discussed further).

Outline

1. Introduction

2. Symmetries

3. Symmetry Breaking
   - Symmetry Breaking by Reformulation
   - Symmetry Breaking by Constraints

4. Dominance Breaking

5. Recap

Symmetries affect solving time

Observation:

- A solver may waste a lot of effort on backtracking over gazillions of (partial) non-solutions that are symmetric to already visited ones
- a found solution can be transformed without search into a symmetric solution in polynomial time (i.e., much faster than searching for them).

Symmetry handling has two aspects:

- Symmetry Detecting: Detect (some) symmetries of the problem or instance or symmetries introduced when modelling.
- Symmetry Breaking: Break (or better exploit) the detected symmetries so that less effort is spent on the solving: multiple symmetric representations of a (non-)solution are avoided.

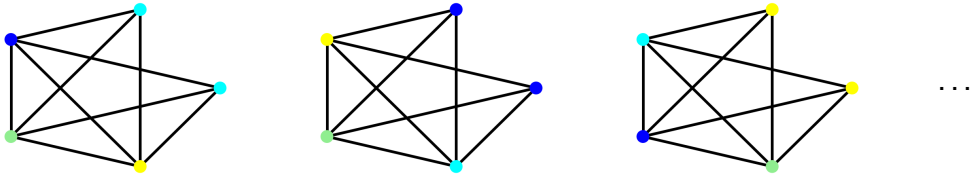
Automated symmetry detection is beyond the scope of this course.

Symmetry Breaking

Definition

A symmetry class is an equivalence class of solutions under all the considered symmetries, including their compositions.

Solutions of the same symmetry class in graph colouring:



...

Aim of Symmetry Breaking: While solving, keep ideally one member (i.e., solution) per symmetry class; this can significantly reduce the search space!

Careful: Size does not matter!

The number of symmetries is no indicator of the difficulty of breaking them!

Symmetry Breaking

Breaking symmetries can be done in different ways.

Types of symmetry breaking:

- Symmetry breaking by reformulation:
  the elimination of the symmetries detectable in the model.
- Static symmetry breaking:
  the elimination of symmetric solutions by adding constraints upfront.
- Dynamic symmetry breaking:
  the elimination of symmetric solutions dynamically during search. (Out of scope)

Symmetry Breaking

Regardless of the type of symmetry breaking used,
our method needs to be sound and efficient:

- Soundness: Symmetry breaking must keep ***at least one*** solution from each symmetry class.

- Efficiency: Symmetry breaking should keep as few solutions as possible from each symmetry class.
  Ideally only 1!

Outline

1. Introduction

2. Symmetries

3. Symmetry Breaking
   - Symmetry Breaking by Reformulation
   - Symmetry Breaking by Constraints

4. Dominance Breaking

5. Recap

Breaking model symmetries by reformulation

Definition (recap)

A model symmetry is a symmetry that arises from a specific representation of a CSP and is not detectable in every model of the problem.

We can break such symmetries by reformulating the model!

Example

Subset problem: Find an n-element subset of a given set S, subject to some constraints.

The model encoding the subset as an array of n decision variables of domain S, constrained to take distinct values, has
$n!$ variable symmetries: the order of the elements does not matter . . .

We can reformulate the model to use boolean variables B, with B_i representing if element i is part of the subset, does not have the variable symmetries of the integer model!

Outline

1. Introduction

2. Symmetries

3. Symmetry Breaking
   - Symmetry Breaking by Reformulation
   - Symmetry Breaking by Constraints

4. Dominance Breaking

5. Recap

Symmetry Breaking by Constraints

Often, not all symmetry can be eliminated by remodelling.

Remaining symmetry should be reduced or eliminated.

- Static symmetry breaking: Using symmetry-breaking constraints.
  - Different than implied constraints, as they change the set of solutions
  - Breaking symmetries with constraints can allow for further implied constraints

- Dynamic symmetry-breaking methods during search. (out of scope)

Adding symmetry-breaking constraints

The added constraints must not exclude any non-symmetric solutions. The goal is to reduce redundancy without affecting the completeness of the search.

- Soundness: Symmetry breaking must keep ***at least one*** solution from each symmetry class.

The added constraints should reduce the number of symmetric assignments that will be explored.

- Efficiency: Symmetry breaking should keep as few solutions as possible from each symmetry class.
  Ideally only 1!

Lexicographic symmetry-breaking constraints

A common way to break symmetries is to impose a lexicographic ordering.

Breaking **variable symmetries**: lexicographic ordering on the variables.

- enforcing an order in the variables, they are no longer indistinguishable and thus not symmetric.
- May also have an effect on value symmetries: (Integer) values have an order that is now important!

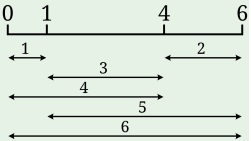
Breaking **index symmetries**: lexicographic ordering on the tensor slices.

- Row symmetries: lexicographic ordering of rows.
- Column symmetries: lexicographic ordering of columns.
- ...

Powerful lexicographic symmetry-breaking global constraints exist in CP.

Lexicographic ordering on variables

Example (Golomb Rulers)



A Golomb ruler has m marks, where all the $\frac{m(m-1)}{2}$ distances between marks ($x_j - x_i \; \forall \; 0 < i, j <= m$) are different

- Variables: one for each mark $x_1, x_2, \ldots, x_m$
- Constraints: $x_j - x_i \neq x_l - x_k$ for all distinct pairs
- **Variable Symmetry**: Variables are symmetrical. Solution $[0, 1, 4, 6]$ is equivalent to $[0, 4, 1, 6]$, and $[1, 0, 4, 6]$, and ...
- Symmetry breaking: Enforce lexicographic ordering with $x_1 < x_2 < \ldots < x_m$

  Or directly use the IncreasingStrict(X) global constraint for symmetry-breaking!

Global constraints for lexicographic ordering of variables

Definition

The INCREASING(X) global constraint holds if and only if the variables in X are in an increasing order:

$$X_i \leq X_j \; \forall \; 0 \leq i < j < |X|$$

Definition

The DECREASING(X) global constraint holds if and only if the variables in X are in a decreasing order:

$$X_i \geq X_j \; \forall \; 0 \leq i < j < |X|$$

Similarly, the constraints INCREASINGSTRICT(X) and DECREASINGSTRICT(X) enforce a strict lexicographic order.

Breaking index symmetries

Example (Latin Squares)

A latin square of size $n \times n$ requires that each row and column contain each number from 1 to n exactly once.

A solution

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

Swap rows 1 and 2

2	3	4	1
1	2	3	4
3	4	1	2
4	1	2	3

Swap columns 1 and 2

2	1	3	4
3	2	4	1
4	3	1	2
1	4	2	3

Lexicographic ordering of lists: $X \leq_{\text{lex}} Y$, with X, Y being lists.

Compare the lists element by element from left to right; the first differing element determines the order.

Slices of tensors are lists, so we can use the $\leq_{\text{lex}}$ operator.

Breaking index symmetries

Example

$[1, 2, 34, 5, 678] \leq_{\text{lex}} [1, 2, 36, 45, 78]$)
because $34 < 36$, even though $678 \not\leq 78$: this $\leq_{\text{lex}}$ order on 1d integer arrays does not require *all* elements to be less than or equal to the corresponding element in the other array; It only considers the first differing pair.

Definition (LexLess() and LexLessEq() global constraints)

The LexLess(X,Y) (resp. LexLessEq(X,Y)) constraint, where X and Y are same-length lists (or 1D arrays) of decision variables, with size n, holds if and only if X is lexicographically less (resp. less or equal) to Y:

- either $n = 0$,
- or $X_1 < Y_1$,
- or $X_1 = Y_1$ & LexLess($[X_i \forall i \in [2..n]], [Y_j \forall j \in [2..n]]$).
  - resp. $X_1 = Y_1$ & LexLessEq($[X_i \forall i \in [2..n]], [Y_j \forall j \in [2..n]]$).

Breaking index symmetries

Lexicographic ordering constraints along one dimension of an array break the index symmetry of that dimension.

Example (Latin Squares)

A latin square of size $n \times n$ requires that each row and column contain each number from 1 to n exactly once.

A solution

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

Swap rows 1 and 2

2	3	4	1
1	2	3	4
3	4	1	2
4	1	2	3

Swap columns 1 and 2

2	1	3	4
3	2	4	1
4	3	1	2
1	4	2	3

Latin squares problem has both row and column symmetries!

Assume variables X_{ij} with $i, j \in [1..n]$. Breaking the row symmetries:

$$\textsc{LexLess}([X_{ij} \ \forall\, j \in [1..n]], [X_{i+1,j} \ \forall\, j \in [1..n]]) \ \forall\, i \in [1..n-1]$$

Lexicographic ordering constraints along every dimension with index symmetry of an array have two properties:

+ No symmetry class is lost. At least one solution remains from each class.
− In general, not all symmetry classes are left with 1 solution, except if the values of the array are distinct, etc.

Counterexample

Assume full row symmetry and full column symmetry:

the arrays

0	0	1
1	1	0

and

0	1	1
1	0	0

have lexicographically ordered rows and columns, but are symmetric: each can be transformed into the other by swapping their two rows as well as swapping their first and last columns.

General scheme for breaking all variable symmetries: The Lex-Leader scheme (see next slide) generates lexicographic ordering constraints that are not necessarily along the dimensions of an array of decision variables.
It guarantees that all the compositions of all variable symmetries are broken.

The Lex-Leader Scheme

Main idea of the Lex-Leader scheme:

- For each equivalence class of solutions under our symmetry group, predefine *one* to be the canonical solution.
- Achieves this by adding constraints that are satisfied by canonical solutions and not by symmetric ones.

For each permutation in a given symmetry group G, Lex-Leader produces one lexicographic constraint.

The set of constraints defined by the Lex-Leader method, using the lexicographic relation $\leq_{\text{lex}}$ in a symmetry group G with a chosen variable ordering V, is

$$\forall \sigma \in G, V \leq_{\text{lex}} \sigma(V)$$

where V is the vector of the variables of the CSP (in a defined order), and $\leq_{\text{lex}}$ is the lexicographic ordering relation

The Lex-Leader Scheme

For any group G of variable symmetries on the indices of the decision variables $x_1, \ldots, x_n$ of domain D, which are not necessarily arranged into a 1d array:

1. Choose an ordering of the decision variables, say $x_1, \ldots, x_n$.
2. Choose a lexicographic-ordering relation, say `lex_lesseq`.
3. For every symmetry $\sigma \in G$, add a constraint enforcing the lexicographic relation

   $$\mathit{LexLessEq}([x_1, \ldots, x_n], [\sigma(x_1), \ldots, \sigma(x_n)])$$

   to the problem model.
4. Simplify the resulting constraints, locally and globally.

This yields exactly one solution per symmetry class.

Example (2×3 array with full row and column symmetry)

Consider the array

x1	x2	x3
x4	x5	x6

with full row and column symmetry: Need 2! (from column symmetry) $\cdot 3!$ (from row symmetry) $-1 = 11$ constraints for the ordering $x_1, x_2, x_3, x_4, x_5, x_6$:

1. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_2, x_1, x_3, x_5, x_4, x_6]$) (for cols 1-2)
2. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_1, x_3, x_2, x_4, x_6, x_5]$) (for cols 2-3)
3. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_4, x_5, x_6, x_1, x_2, x_3]$) (for rows)
4. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_6, x_4, x_5, x_3, x_1, x_2]$) (for rows + cols 2-3, 1-2)
5. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_5, x_6, x_4, x_2, x_3, x_1]$) (for rows + cols 1-2, 2-3)
6. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_4, x_6, x_5, x_1, x_3, x_2]$) (for rows + cols 2-3)
7. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_5, x_4, x_6, x_2, x_1, x_3]$) (for rows + cols 1-2)
8. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_6, x_5, x_4, x_3, x_2, x_1]$) (for rows + cols 1-3)
9. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_3, x_2, x_1, x_6, x_5, x_4]$) (for cols 1-3)
10. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_2, x_3, x_1, x_5, x_6, x_4]$) (for cols 1-2, 2-3)
11. LexLessEq($[x_1, x_2, x_3, x_4, x_5, x_6], [x_3, x_1, x_2, x_6, x_4, x_5]$) (for cols 2-3, 1-2)

Symmetry Breaking

The Lex-Leader Scheme is general, but takes exponential space if there are exponentially many symmetries!

Problem-specific symmetry-breaking constraints may be more beneficial.

Note: Breaking all the symmetries may increase the solving time:

- More propagation time than reduction in search time
- Break only some symmetries, but which ones?

Often problem and model specific.

Outline

1. Introduction

2. Symmetries

3. Symmetry Breaking
   - Symmetry Breaking by Reformulation
   - Symmetry Breaking by Constraints

4. Dominance Breaking

5. Recap

Dominance Breaking

- Symmetry breaking can be beneficial also for optimization problems (for any symmetric solutions s_1, s_2 we have $f(s_1) = f(s_2)$);
- But we can *generalize* the concept of symmetry breaking in optimization problems.
- In optimization problems, the *minimization* of an objective function f determines optimality; exploring suboptimal assignments should be avoided.
- Many constraint problems exhibit dominance relations which can be exploited for dramatic reductions in search space.

Definition

Given a problem with an objective function f, a dominance relation $\prec$ is a binary relation on the set of solutions S, such that if $s_1 \prec s_2$ for $s_1, s_2 \in S$, then one of the following is true:

1. s_1 is a solution and s_2 is not.
2. They are both solutions and $f(s_1) < f(s_2)$.
3. They are both non-solutions and $f(s_1) < f(s_2)$.

Dominance Breaking

Main idea: A (partial) assignment that satisfies the constraints can be forbidden if it is dominated.

- for any solution that this assignment would lead to, there must be another solution that is equally good or better

Identify dominance relations in the problem that can be used to cut the search space.

Add dominance-breaking constraints to reduce the search space and speed up the solving process.

Example (Dominance)

Consider a simple problem with:

- Domain: $x_i \in \{1, \ldots, 10\}$,
- Constraint: $\textsc{AllDifferent}([x_1, \ldots, x_{10}])$,
- Objective function: $\sum_{i=1}^{10} i \cdot x_i$ to be minimized.

The partial assignment $s_1 = [x_1 = 2, x_2 = 1]$ **dominates** the partial assignment $s_2 = [x_1 = 1, x_2 = 2]$, because:

No matter how we label the remaining variables, the corresponding complete assignments s'_1, s'_2 with same values for $x_3, \ldots, x_{10}$, will always have $f(s'_1), < f(s'_2)$.

This is due to:

$$1 \cdot 2 + 2 \cdot 1 < 1 \cdot 1 + 2 \cdot 2.$$

Example (Dominance Breaking)

Identify the dominance relation in the previous example:

We have the objective function $f = \sum_{i=1}^{10} i \cdot x_i$.

To minimize the objective, we want higher values for variables with smaller indices because so that they have less impact on the total sum.

The following dominance exists for this objective, without contradicting the constraint

$$x_i \geq x_j, \ \forall i < j$$

Due to the AllDifferent() constraint, we can simplify it to

$$x_i > x_j, \ \forall i < j$$

By enforcing such a **lexicographic order** in our variables, dominated assignments will not be explored during the search process

Outline

1. Introduction

2. Symmetries

3. Symmetry Breaking
   - Symmetry Breaking by Reformulation
   - Symmetry Breaking by Constraints

4. Dominance Breaking

5. Recap

Recap

Often symmetries exist in the problem, wasting search effort

- Variable and value symmetries.

Different types based on where symmetries are introduced:

- problem symmetries are detectable in *every* instance and model of the problem.
- instance symmetries are detectable in *specific instances* of a problem.
- model symmetries are *not* detectable in every model, but only in specific viewpoints.

Symmetry breaking is a very efficient method to speed up the solving process

Powerful lexicographic global constraints exist to break symmetries

The Lex-Leader scheme offers a general method to break symmetries

Recap

- Keep in mind the objective: first solution, all solutions, or best solution? Symmetry breaking might pay off more in some cases (Unsat, all solutions, optimal solution)

- Problem constraints can sometimes be simplified in the presence of symmetry-breaking constraints.
  Example: $z = \text{ABS}(x - y)$ can be simplified into $z = x - y$ if symmetry-breaking requires $x \geq y$; $z = x - y$ is much easier for solvers than $z = \text{ABS}(x - y)$

- Dominance breaking offers additional improvements for optimization problems!