

Automatic Symmetry Detection

- For CSP instances
 - Many different methods exist.
 - + Trade-off between speed and completeness.
 - Flexible: detect different kinds of symmetries.
 - Accurate: detect many or all symmetries.
 - The more powerful are impractical for large instances.
 - Results are instance specific.
- For CSP models
 - Roy and Pachet, ECAI98; Van Hentenryck et al, SARA05
 - + Results apply to all instances.
 - Requires modelling with global constraints.
 - Detect only certain kinds of symmetries.
 - Can easily lose accuracy.

Aim

The aim of our work is to develop a method of automatic symmetry detection that:

- operates on models rather than instances,
- is as independent of the syntax as possible,
- can detect as many symmetries as possible,
- is practical.

Our Framework

- 1 Find the symmetries of several small instances.
- 2 Generalise these symmetries.
- 3 Filter the symmetries to produce some candidate symmetries.
- 4 Prove (or disprove) that the candidates hold for the model.

Next: a simple implementation of the first three steps.

Step 1: Find Instance Symmetries

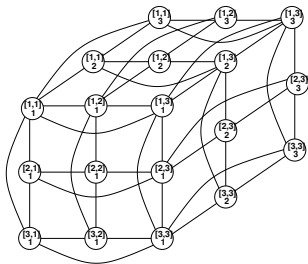
- Find the symmetries of some instances,
 - ... but which instances?
- Assume models are parametrised by n integer parameters.
- Increase each parameter separately.
- N-queens: $N = 4, N = 5, N = 6$ (3 instances)
- $S \times S$ Latin square: $S = 3, S = 4, S = 5$ (3 instances)
- Social golfers: (Weeks, Groups, Size) =
 - (2,2,2), (3,2,2), (4,2,2)
 - (2,2,2), (2,3,2), (2,4,2)
 - (2,2,2), (2,2,3), (2,2,4) (7 instances)

Step 1: Find Instance Symmetries (cont.)

- Obtain one instance per parameter.
- Find the symmetries of each instance.
- Use your favourite method.
- You could use, for example:
 - **Our method (Symcon06)**
 - Puget's method (CP05)
 - The microstructure complement (Cohen et al, CP05)
- Each of the methods listed above uses graph automorphisms.

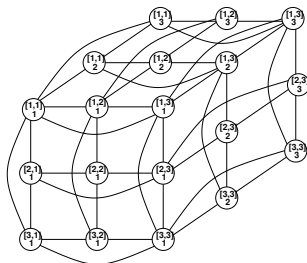
Latin Square Example

N=3

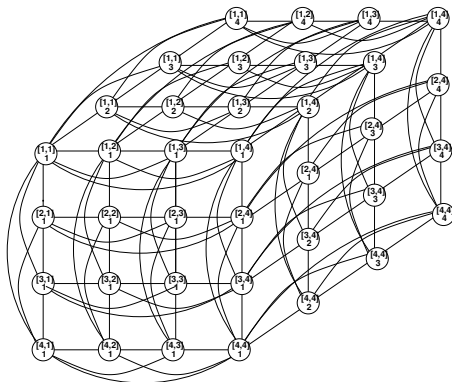


Latin Square Example

N=3



N=4



Latin Square (3) Generators

(Board[1, 2]=1, Board[1, 3]=1) (Board[1, 2]=2, Board[1, 3]=2)
 (Board[1, 2]=3, Board[1, 3]=3) (Board[2, 2]=1, Board[2, 3]=1)
 (Board[2, 2]=2, Board[2, 3]=2) (Board[2, 2]=3, Board[2, 3]=3)
 (Board[3, 2]=1, Board[3, 3]=1) (Board[3, 2]=2, Board[3, 3]=2)
 (Board[3, 2]=3, Board[3, 3]=3)

(Board[2, 1]=1, Board[3, 1]=1) (Board[2, 1]=2, Board[3, 1]=2) (Board[2, 1]=3, Board[3, 1]=3)
 (Board[2, 2]=1, Board[3, 2]=1) (Board[2, 2]=2, Board[3, 2]=2) (Board[2, 2]=3, Board[3, 2]=3)
 (Board[2, 3]=1, Board[3, 3]=1) (Board[2, 3]=2, Board[3, 3]=2) (Board[2, 3]=3, Board[3, 3]=3)

(Board[1, 2]=1, Board[2, 1]=1) (Board[1, 2]=2, Board[2, 1]=2) (Board[1, 2]=3, Board[2, 1]=3)
 (Board[1, 3]=1, Board[3, 1]=1) (Board[1, 3]=2, Board[3, 1]=2) (Board[1, 3]=3, Board[3, 1]=3)
 (Board[2, 3]=1, Board[3, 2]=1) (Board[2, 3]=2, Board[3, 2]=2) (Board[2, 3]=3, Board[3, 2]=3)

(Board[1, 1]=2, Board[1, 1]=3) (Board[1, 2]=2, Board[1, 2]=3) (Board[1, 3]=2, Board[1, 3]=3)
 (Board[2, 1]=2, Board[2, 1]=3) (Board[2, 2]=2, Board[2, 2]=3) (Board[2, 3]=2, Board[2, 3]=3)
 (Board[3, 1]=2, Board[3, 1]=3) (Board[3, 2]=2, Board[3, 2]=3) (Board[3, 3]=2, Board[3, 3]=3)

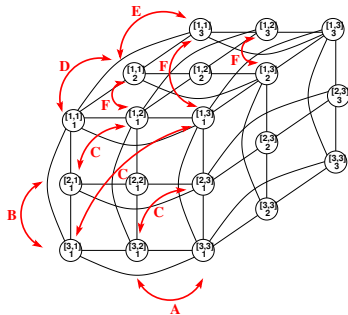
(Board[1, 1]=2, Board[1, 2]=1) (Board[1, 1]=3, Board[1, 3]=1) (Board[1, 2]=3, Board[1, 3]=2)
 (Board[2, 1]=2, Board[2, 2]=1) (Board[2, 1]=3, Board[2, 3]=1) (Board[2, 2]=3, Board[2, 3]=2)
 (Board[3, 1]=2, Board[3, 2]=1) (Board[3, 1]=3, Board[3, 3]=1) (Board[3, 2]=3, Board[3, 3]=2)

(Board[1, 1]=1, Board[1, 1]=2) (Board[1, 2]=1, Board[1, 2]=2) (Board[1, 3]=1, Board[1, 3]=2)
 (Board[2, 1]=1, Board[2, 1]=2) (Board[2, 2]=1, Board[2, 2]=2) (Board[2, 3]=1, Board[2, 3]=2)
 (Board[3, 1]=1, Board[3, 1]=2) (Board[3, 2]=1, Board[3, 2]=2) (Board[3, 3]=1, Board[3, 3]=2)

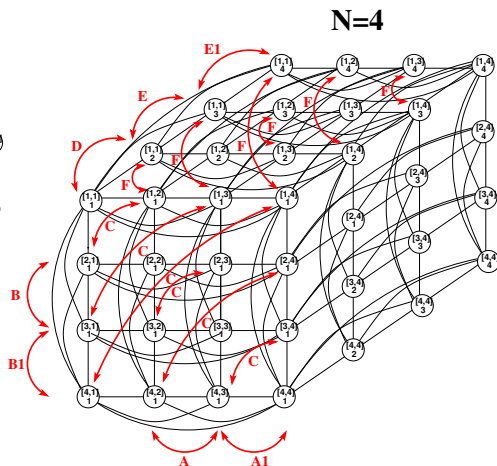
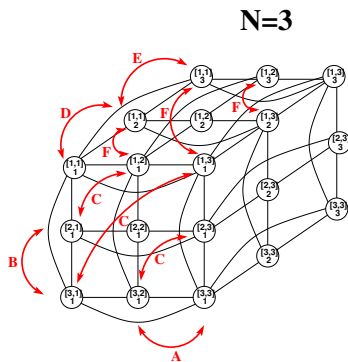


Latin Square Example

$N=3$



Latin Square Example



A Word on Group Generators

- Many instance methods produce symmetry generators.
- Our method uses Saucy to find automorphisms.
- Saucy produces a set of group generators.
- We can deal with the generators themselves, or use a tool such as GAP to deal with the group as a whole.

Step 2: Generalise each Generator

- We have the symmetries (or generators) relative to each instance.
- We want to express the generators such that they are independent of the particular instance.
- That is, we want to “lift” the generators to the model.
- Classify each generator by a pattern that describes a model symmetry.

Step 2: Generalise each Generator (cont.)

- Consider the variables and values as a matrix.
- Classify generators into simple categories.
 - Two values swapped in one dimension (row/column swap).
 - The values of a dimension inverted (matrix reflection).
 - Two dimensions swapped (diagonal matrix reflection).
- Ignore generators that can't be so classified.
- Simple pattern matching is inherently incomplete.
- Instead of pattern matching, machine learning could be used for intelligent pattern recognition.

Latin Square (3&4) Generalised Generators

- Dimension 1: swap values 2 and 3
- Dimension 2: swap values 2 and 3
- Dimension 3: swap values 2 and 3
- Dimension 3: swap values 1 and 2
- Dimension 1 and 2 interchange
- Dimension 2 and 3 interchange

- Dimension 1: swap values 2 and 3
- Dimension 1: swap values 3 and 4
- Dimension 2: swap values 2 and 3
- Dimension 2: swap values 3 and 4
- Dimension 3: swap values 1 and 2
- Dimension 3: swap values 2 and 3
- Dimension 3: swap values 3 and 4
- Dimension 1 and 2 interchange
- Dimension 2 and 3 interchange

Step 2: Generalise each Generator (cont.)

- This amount of generalisation is sufficient for some cases (e.g. N-queens).
- Some symmetries benefit from being composed, for example:
 - Value-swaps in a dimension can be merged.
 - At best, all the values in the dimension are interchangeable.
- Pros: Simple and fast.
- Cons: Incomplete; possible improvement: machine learning.

Latin Square (3&4) Generalised Generators Again

- Dimension 1: swap values 2 and 3
 - Dimension 2: swap values 2 and 3
 - Dimension 3: all values interchangeable
 - Dimension 1 and 2 interchange
 - Dimension 2 and 3 interchange
-
- Dimension 1: values 2, 3 and 4 interchangeable
 - Dimension 2: values 2, 3 and 4 interchangeable
 - Dimension 3: all values interchangeable
 - Dimension 1 and 2 interchange
 - Dimension 2 and 3 interchange

Step 3: Determine Candidate Symmetries

- Having gathered all the generalised symmetries, we propose candidates for the model.
- One simple way to suggest good candidates is to take those that are present in every examined instance.
- Finds all symmetries for:
 - N-queens (integer- and boolean-versions)
 - Latin Square
- However, some good candidates may be missed by this method due to inconvenient generator sets given by Saucy.

Candidates for Latin Square

These generalised generators are present in the Latin Square instances of size 3, 4 and 5.

- Dimension 1 and 2 interchange
- Dimension 2 and 3 interchange
- Dimension 3: all values interchangeable

Symmetry Inference

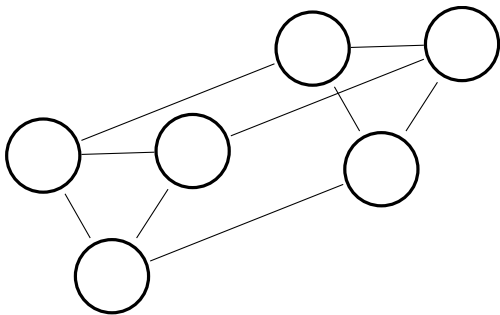
- In the 3x2 graceful graph instance, Saucy produces a generator that doesn't match any pattern.
- However, the “correct” symmetries are found for the 4x2 and 5x2 instances:
 - Dimension 1: all values interchangeable
 - Dimension 2: values invertible
 - Dimension 3: values invertible
- We can query the group for the 3x2 instance to see whether these symmetries are really there.

Step 4: Proving symmetries of the model

- The final step is to determine whether each candidate is a true symmetry of the model.
- We don't propose a new method for this step yet, but some work has already been done (e.g. Mancini and Cadoli, 2005).
- Such theorem-proving methods are inherently incomplete.
- We would prefer a method based on the construction of the graph (step 1); this is future work.

Another Example

$K_n \times P_m$ Graceful Graphs.



$K_3 \times P_2$

Instances for Step 1

- $(3, 2) : K_3 \times P_2$
- $(4, 2) : K_4 \times P_2$
- $(5, 2) : K_5 \times P_2$

- $(3, 2) : K_3 \times P_2$
- $(3, 3) : K_3 \times P_3$
- $(3, 4) : K_3 \times P_4$

Instance Symmetries

$$K_3 \times P_2$$

- Dimension 1: all values invertible
- Dimension 1: swap values 1 and 2
- Dimension 3: all values invertible
- **unmatched generator**

$$K_4 \times P_2$$

- Dimension 1: all values interchangeable
- Dimension 2: all values invertible
- Dimension 3: all values invertible

$$K_5 \times P_2$$

- Dimension 1: all values interchangeable
- Dimension 2: all values invertible
- Dimension 3: all values invertible

Instance Symmetries (cont.)

$$K_3 \times P_2$$

- Dimension 1: all values invertible
- Dimension 1: swap values 1 and 2
- Dimension 3: all values invertible
- **unmatched generator**

$$K_3 \times P_3$$

- Dimension 1: all values interchangeable
- Dimension 2: all values invertible
- Dimension 3: all values invertible

$$K_3 \times P_4$$

- Dimension 1: all values interchangeable
- Dimension 2: all values invertible
- Dimension 3: all values invertible

Instance Symmetries (cont.)

$$K_3 \times P_2$$

- Dimension 1: all values invertible
- Dimension 1: swap values 1 and 2
- Dimension 3: all values invertible
- unmatched generator

$$K_3 \times P_3$$

- Dimension 1: all values interchangeable
- Dimension 2: all values invertible
- Dimension 3: all values invertible

$$K_3 \times P_4$$

- Dimension 1: all values interchangeable
- Dimension 2: all values invertible
- Dimension 3: all values invertible

Candidate Symmetries

- Dimension 1: all values interchangeable
- Dimension 2: all values invertible
- Dimension 3: all values invertible

Results

- We have tried our simple implementation on the following models:
 - N-queens (all)
 - Latin Square (all)
 - Graceful Graph $K_n \times P_m$ (all via group)
 - NxN queens (all via group)
 - Steiner Triple (all via group)
 - Social Golfers (most via group)
 - Golomb ruler (no pattern match)

Properties of Our Framework

- Flexibility: replaceable parts
- Practicality: can compromise completeness for speed
- Effectiveness: can find many useful symmetries

Achievements, Limitations and Future Work

- Achievements
 - A radically new framework for detecting model symmetries
 - Less dependent on syntax of model
 - An implementation that gives good results
- Limitations
 - The completeness and practicality of the approach depends on its parts
- Future work
 - A better generalisation step
 - A proof step based on the graph construction