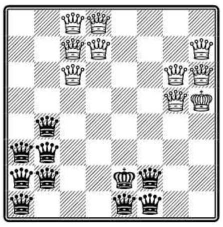
A chess puzzle



- 9 queens and 1 king of each colour
- no piece on the same line as a queen of the opposite colour
- unique up to symmetry
- found by SBDS in ILOG Solver
- Smith, Petrie & Gent, 04
- and it's a legal chess position!

Symmetry in Constraint Programming

Ian Gent St Andrews

Symmetry in Constraint Programming

Ian Gent St Andrews

Special thanks to …

Jean-François Puget ILOG, France

Special thanks to

Karen Petrie	Oxford
Barbara Smith	Leeds
Ian Miguel	St Andrews
Alan Frisch	York
Colva Roney-Dougal	St Andrews
Steve Linton	St Andrews
Sarah Gent	Castlehill School

"A project of this scope and importance could not be achieved without the aid and assistance of many people…

*…. or rather it **could** but it would be dumb to do it that way when there are so many people around willing to give their aid."*

- Peter Schickele

Shameless self promotion

- A most excellent survey has recently been written….
- "Symmetry in Constraint Programming"
  - Ian Gent, Karen Petrie, Jean-François Puget
  - *In "Handbook of Constraint Programming"*
    - *Ed. Rossi, van Beek, & Walsh*
    - *Elsevier, 2006*
  - I can give you a copy if you are interested
- Error in paper: (noticed by Pedro Meseguer)
  - Example 10.2 p 332
  - Cyclic forms of r90 and r270 are wrong.
  - r90 should be (1 7 9 3) (2 4 8 6)
  - r270 should be (1 3 9 7) (2 6 8 4)

Summer Schools are Great…

- They can lead to untold happiness
  - Joy
  - Delight
- You think I'm kidding?
- How did a British person meet an American in Germany in 1991?

How to get here: Giving Lectures at Summer School

1. Do a PhD in something else.
2. Go to a summer school in that.
3. Follow the stupidest dating strategy in the world (for a man).
4. Change area and do something completely different.
5. Get into constraints.
6. Work on something completely different
7. Write a paper on symmetry.
8. Think that there is not much future in the area.
9. Work with people smarter than you.
10. Write lots of papers with them.

Free advice

- Dotted throughout lectures will be some free advice on being an academic
- Well it's not really free
  - Since you paid to be here
- And you might have seen it before
  - In doctoral programme of CP 06
- And it may be rubbish
  - Check out my dating advice
- ***BUT*** *without any question at all*
  - It is advice.

Best free advice I can give

- Work with people smarter than you.
  - Be the stupidest person you work with!
- Most people are scared of working with smarter people.
  - They feel good if they are smarter than their colleagues.
- Think of it this way:
  - Most computer scientists collaborate.
  - The smarter the group the better the output tends to be.
  - You can't make yourself more intelligent quickly.
  - Find the smartest people you can who don't think it's a waste of their time working with you.
- Or think of it this way:
  - Will your paper be better if your co-authors are
  - *smarter* than you or
  - *stupider* than you?
- P.s. I applied this to my choice of wife as well :-)

Always be the stupidest person you work with!

- So if you constantly feel you are the stupidest one in the room
  - Try to feel good about it!
  - I should know ….
    - Since I've worked with …
    - Joe Culberson, Jeremy Frank, Enrico Giunchiglia, Warwick Harvey, Holger Hoos, Chris Jefferson, Tom Kelsey, Steve Linton, Inês Lynce, Ian Miguel, Massimo Narizzano, Pete Nightingale, Karen Petrie, Patrick Prosser, Andrea Rendl, Colva Roney-Dougal, Andrew Rowley, Kostas Stergiou, Paul Shaw, Barbara Smith, Toby Walsh, …
    - … and Judith Underwood.

Section 10.1, p331

Group Theory

- What is a group?
- Standard mathematical answer is:
  - Group = pair of set and operation <G,*> with
    1. For all g,h in G, g*h in G
    2. There is 1 in G such that for all g in G, 1*g = g*1 = g
    3. For all g in G, there is g^{-1} such that $g*g^{-1} = g$
    4. For all g,h,j in G, (g*h)*j = g*(h*j)
  - What has this got to do with symmetry?
- Ordinary answer:
  - A group is an organisation of people.
  - Unfortunately not the answer we need!
- What's the link between groups and symmetries?

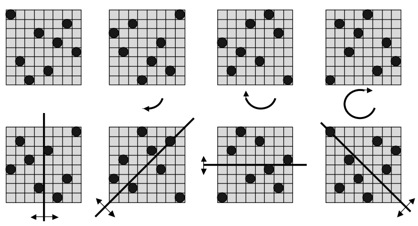
Group Theory

- Let's start again.
- Mathematics deals with abstractions
- E.g.
  - Abstraction of repetition ➜ Integers
  - Abstraction of continuous quantities ➜ Real numbers
- Similarly
  - Abstraction of symmetry ➜ Group Theory

Group Theory

- I want to explain tiny bits of group theory
- Explaining relevance to Constraints
  - Eventually we'll see why group axioms hold
  - Introduce a few key concepts that come up
- I am going to focus on
  - The ***action*** of group elements
    - Not usually focus of simple introductions
    - But what constraint programmers need to think about
  - Group elements as ***permutations***

8 symmetries of the square



Rotation though 90 degrees: r90

1	2	3
4	5	6
7	8	9

$\longrightarrow$

7	4	1
8	5	2
9	6	3

- Two ways to think about this
  - 1 moves to 3, 2 moves to 6 …
  - 1 *is replaced by* 7, 7 is replaced by 9 …
- In the paper we used the second version
  - sometimes getting it wrong, sorry.

This defines an **action** on **points**

1	2	3
4	5	6
7	8	9

$\longrightarrow$

7	4	1
8	5	2
9	6	3

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

- This is a ***permutation***
- Each point is mapped to exactly once
- Above is written in *Cauchy form*

This defines an **action** on **points**

1	2	3
4	5	6
7	8	9

$\longrightarrow$

7	4	1
8	5	2
9	6	3

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

- Presentations more normally written in **cycle form**
- (1 7 9 3) (2 4 8 6) (5)
- (1 7 9 3) (2 4 8 6)

Where have we got to?

- A symmetry is a mapping of an object
  - E.g. r90 is a function on a square
- Leaves some features unchanged
  - It is still a 3x3 square
- We label the object with points
  - The Symmetry possibly changes those points
  - E.g. 1 is replaced by 7
- A permutation summarises the changes made
  - Where each point is moved to
  - The permutation is a permutation of the points
- We say that the symmetry *acts* on the points
  - E.g. $1^{r90} = 7$
  - Action can be extended to sets of points
    - E.g. $\{1,3,8\}^{r90} = \{1^{r90}, 2^{r90}, 3^{r90}\} = \{7,1,6\} = \{1,6,7\}$

Composition of Permutations

- Two symmetries acting in a row *compose*
- Permutations compose in the same way
  - E.g.
  - r90 = (1 7 9 3) (2 4 8 6)
  - x = (1 3)(4 6)(7 9) (reflection about centre col)
  - r90 * x = (1 9)(2 6)(4 8)
- **NOTE:** in r90 * x, r90 is done first
  - $1^{r90 * x} = (1^{r90})^{x} = 7^{x} = 9$

Identity and inverse

- Permutations make identity and inverse easy
- Identity:
  - () in cycle form
  - In Cauchy form

  1 2 3 4 5 6 7 8 9
  1 2 3 4 5 6 7 8 9
- Inverse
  - Just reverse each (..) in cycle form

  r90 = (1 7 9 3) (2 4 8 6)
  $r90^{-1}$ = (3 9 7 1)(6 8 4 2) = (1 3 9 7)(2 6 8 4) = r270

Associativity

- (f * g) * h = f * (g * h)
- Basically function application always respects this
- Doing (f and then g) and then h
  - Is the same as doing f and then (g and then h)
- Because of the way we thought about composition, there is nothing to say

Group Axioms again

- Group = pair of set and operation <G,*> with
  1. For all g,h in G, g*h in G
  2. There is 1 in G such that for all g in G, 1*g = g*1 = g
  3. For all g in G, there is g^{-1} such that $g*g^{-1} = g$
  4. For all g,h,j in G, (g*h)*j = g*(h*j)
- G = set of permutations
  - E.g. {1, r90, r180, r270, x, y, d1, d2}
  - **See?** *Elements* of the group are themselves functions. They act on the points in our square
- * = composition of permutations
  1. Composition of two permutations is another permutation
  2. () is the identity permutation
  3. We've seen how to get the inverse
  4. Associativity is fundamental

Generators

- Number of elements is ***order*** of the group
- Closure and inverse lets us ***generate*** a group from a small number of elements.
  - E.g. chessboard from {r90,x}
  - The maximum size of a (minimal) generating set for a group of order n is $\log_2 n$
  - Typically groups of arbitrary size can be generated by 2 elements!
- Use of generators fundamental in *computational algebra*
  - Can save exponential space
  - Can be vital for time as well
    - Don't want to step through exponential elements.

Moving right along

- Going to mention a few things briefly
- ***Subgroup***
  - <H,*> a subgroup of <G,*> if H subset of G and H closed under *
- ***Coset***
  - If H is a subgroup of G and g any element of G
  - Then a *left coset* g * H = { g * h | h in H}
    - Two cosets g * H, f * H are equal or disjoint
    - All cosets are of size | H |
      - Hence Lagrange's theorem
      - order of subgroup divides order of group
  - A set of ***coset representatives*** is a set R subset of G
    - Such that, for all g in G, g*H = r*H for some r in R

Orbit & Stabiliser

- The ***orbit*** of a point p in group G is
  - $p^G = \{p^g \mid g \text{ in } G\}$
  - I.e. everywhere a point can be moved to by elements of G
- The ***stabiliser*** of a point p in G is
  - $G_p = \{g \mid g \text{ in } G \text{ and } p^g=p\}$
  - I.e. set of points that fixes p
  - Stabiliser is always a subgroup of G.

Section 10.1.2, p337

Schreier Sims

- Key algorithm in computational algebra
  - Akin to Arc Consistency in Constraint Satisfaction
  - You may need to implement it one day
- Computers a ***stabiliser chain*** and a set of *coset representatives* at each level
  - Say we have listed points in order 1, 2, 3 … n
    - $G_0 = G$
    - $G_1 = (G_0)_1$
    - $G_2 = (G_1)_2$
    - …
    - $G_i = (G_{i-1})_I$
    - …
    - G_n = {1} (since only identity fixes every point)
  - Schreier Sims computes the stabiliser chain efficiently *AND* gives a data structure storing this and the set of coset representatives at each level.

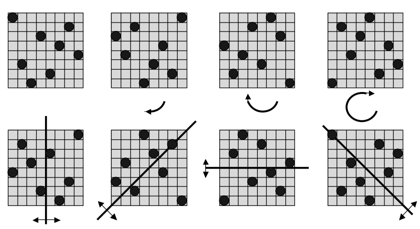
Tip

Never read reviews the day your paper is accepted or rejected

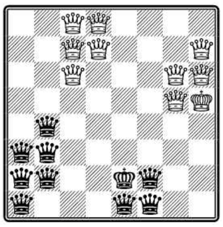
Never read reviews the day your paper is accepted or rejected

- About 15 years ago Alan Bundy said …
  - … getting reviews always ruined his day
  - Even for accepted papers!
- For accepted papers?
  - One reviewer is negative
  - One is lukewarm
  - One reviewer misses the point completely
  - They suggest changing the title you love
- Get over the news first
  - Enjoy it, celebrate, look forward to a trip, consider how clever you must be to publish in International Symmetry Conference
  - Get over it, get on with your life, consider how lucky you are not to have to go to the International Symmetry Conference, what with stupid flights and AAAI next year is a much better conference
- Then read the reviews the next day

8 symmetries of the square



A chess puzzle



- 9 queens and 1 king of each colour
- no piece on the same line as a queen of the opposite colour
- unique up to symmetry
- found by SBDS in ILOG Solver
- Smith, Petrie & Gent, 04
- and it's a legal chess position!

Symmetry Breaking in Constraint Programming

- Many constraint problems have symmetry
  - n-queens, colouring, golfers' problem, …
- Breaking symmetry reduces search
  - avoids exploring equivalent states
  - not sure if "breaking symmetry" is right term, but we're stuck with it
  - in fact preferring (using some method) a subset (preferably singleton) of each equivalence classe
- Note that main goal is ***pragmatic***
  - make constraint programming more effective

Symmetries

- Isomorphisms
  - 1-1 Mappings (bijections) that preserve problem structure.
  - Variables can be permuted
  - Values can be permuted
  - Both
- Map solutions to solutions
  - Potentially large number of isomorph variants
- Map trees search to tree search
  - The same failure will be repeated many times

Symmetry Definitions

- Not really sorted out until 2005!
  - Cohen, Jeavons, Jefferson, Petrie, Smith
  - What was going on until then?
- Can think of symmetries semantically or syntactically
- **Solution Symmetry**
  - A permutation of the variable-value pairs which leaves the set of solutions unchanged
- **Problem Symmetry**
  - A permutation of the variable-value pairs which leaves the set of constraints unchanged
- All problem symmetries are solution symmetries, but not vice versa
  - Cohen et al go into more detail with interesting results

Variable and Value symmetries

- **Variable Symmetry**
  - A symmetry which only changes variables
  - E.g. $(x=1)^g = (y=1)$ leaves the value 1 unchanged
- **Value Symmetries**
  - A symmetry which only changes values
  - E.g. $(x=1)^g = (x=7)$ leaves the variable x unchanged
- Various pieces of research in constraints focus on one or the other.
  - Sometimes for fundamental reasons
  - Sometimes for convenience

Constraints and Group Theory

- Typically, points = set of variable-value pairs
  - E.g. if n variables with m values, nm points.
  - Sometimes only n if dealing with variable symms.
  - Or only m if dealing with value symms
- Permutations act on these points (v-v pairs)
  - Set of permutations defines the group

Section 10.8, p365

Getting Symmetry in the System

- How does a constraint program know about symmetry?
- Most work assumed symmetries given by the programmer
  - Which isn't realistic in practice
- Attempts to get simple language to express symmetries
  - E.g. in ECLiPSe symmetry libraries
- Significant efforts on *detecting* symmetries
  - Almost inevitably problem symmetries, not solution ones
  - Use graph of some representation of the problem
  - Use graph automorphism software on the result
    - E.g. Aloul *et al* in SAT, Puget in CP

Symmetry Breaking in Constraint Programming

- Three main approaches to symmetry breaking
  - reformulate the problem
  - adapt search algorithm to break symmetry
  - add constraints before search

Symmetry Breaking in Constraint Programming

- Three main approaches to symmetry breaking
  - **reformulate the problem**
  - adapt search algorithm to break symmetry
  - add constraints before search

Section 10.3, p340

Symmetry Breaking by Reformulation

- Reformulation can be the most effective means to break symmetry
- As yet, there is very little general to say about it
  - general methods and/or theorems would be welcome
- This is a common feature in AI
  - we know problem representation is vital
  - we don't know how to exploit it except by magic

An example

- Crew assignment
  - One variable per crew staff : x[j]
  - One value per crew : a[i]
  - A size per crew : c[i]

x in a;
distribute(x,card,a);
forall (i) card[i] <= c[i];

- Add

x[1] <= x[2] <= … <= x[n] breaks variable permutations
card[1] >= card[2] >= … >= card[n] breaks value permutations

Problem reformulation

- Use set variables if possible
- For the crew example, use

A set variable s[i] per crew
A value per staff.

  - This breaks permutations of people
- Add

```
card(s[1]) >= card(s[2]) >= … >= card(s[n])
```

  - This breaks permutations of crews

Reformulation example

- All Interval Series Problem
- Write down the numbers 0…n-1
  - so that each difference 1...n-1 occurs between consecutive terms
  - e.g. 0 8 1 7 2 6 3 5 4
  - diff   8 7 6 5 4 3 2 1
  - now count number of solutions
- Symmetries: reverse sequence & complementation
- This problem is not important but there is a dramatic reformulation

Reformulation example

- Reformulated All Interval Series Problem
- write down the numbers 0…n-1 ***in a cycle***
  - so that each difference 1...n-1 occurs between consecutive terms ***and one difference occurs exactly twice***
  - e.g. 0 8 1 7 2 6 3 5 4
  - diff   8 7 6 5 **4** 3 2 1 **4**
  - now count number of solutions
- Symmetries: reverse sequence & complementation
  - **& rotation of sequence by j steps**
- Each solution yields two solutions to the original
  - e.g. 6 3 5 4 0 8 1 7
- We can break ***all*** symmetry very easily
  - set first 3 terms to be 0 n-1 1
- Improved the state of the art by a factor of 50 in run time

Tip

Do read negative reviews

Do read negative reviews

- There is always the amazing possibility that the person who wrote the review…
  - … is ***not*** an idiot
- Even if they are an idiot…
  - … there may be other idiots out there
  - … who will review your future papers!
- Try to make the changes they want
  - Correct typos, redo experiments, clarify explanations…
- Look out for patterns in reviews
  - You might be making the same mistake again and again
- After all this *you* will think your papers are better

Symmetry Breaking in Constraint Programming

- Three main approaches to symmetry breaking
  - reformulate the problem
  - **add constraints before search**
  - adapt search algorithm to break symmetry

Section 10.4, p343

Symmetry Breaking Constraints

- Probably the grandmother of symmetry breaking constraints
- Added ad hoc since the beginning of time
- e.g.
  - X <= Y <= Z … if S_n acts on variables
  - the first queen is to the left of the second queen
- Difficult to be sure you have eliminated all symmetry
- Requires considerable insight from programmer
- Some symmetries require large constraints
- But easy for constraint programming systems to cope with

Section 10.4.1, p343

Lex-Leader

- Crawford, Ginsberg, Luks & Roy, 1996
  - biggest single advance in symmetry breaking in SAT & Constraints?
- Idea essentially simple
- Define a canonical solution and add constraints to choose it

- Note: technically only applies to variable symmetries. Recently extended by Puget and Walsh (separately, CP06).

Canonical solutions

- Assume an ordering on variables
- Define an ordering on solutions
- How to compare two solutions a and b?
- Use a lexicographic ordering:
  - Write the values in the order given by the variable ordering
  - Compare the two numbers obtained this way
- A canonical solution s is smaller than any of its symmetrical variant
  - S ≤Lex g(s), for all symmetries g.

Example: a 2x3 Matrix

- E.g. swap rows and first and last column
- ABCDEF <= FEDCBA
- There are 12 elements in the symmetry group

A	B	C
D	E	F

F	E	D
C	B	A

Example: a 2x3 Matrix

A	B	C
D	E	F

1. ABCDEF <= ABCDEF
2. ABCDEF <= ACBDFE
3. ABCDEF <= BACEDF
4. ABCDEF <= CBAFED
5. ABCDEF <= BCAEFD
6. ABCDEF <= CABFDE
7. ABCDEF <= DEFABC
8. ABCDEF <= DFEACB
9. ABCDEF <= EDFBAC
10. ABCDEF <= FEDCBA
11. ABCDEF <= EFDBCA
12. ABCDEF <= FDEABC

Lex-Leader

- One constraint for each symmetry
- Just like SBDS, and just as unscaleable
- So we have to choose subsets of symmetries
- Most research now is (or can be seen as) making sensible choices of subsets
- Sensible means
  - useful for commonly occurring symmetry groups
  - amenable to efficient implementation
- Usually lose completeness of symmetry breaking
  - except in some special cases

Section 10.4.4, p348

State some lex leader constraints

- State lex leader constraints for some symmetries, not all
- State for generators only (Aloul et al)
- State for row and column swaps in matrix models
  - Double lex
- State for symmetries not yet broken
  - STAB

Matrix models

- Many problems can be modelled by matrices of decision variables.
  - E.g. Configuration, Scheduling, Design Problems:
  - Rack Configuration.
  - Social Golfers.
  - Steel Mill Slab Design.
- Frequently these matrices exhibit row and/or column symmetry.

Example: a 2x3 Matrix

A	B	C
D	E	F

- Lex leader for swap adjacent columns Ci, Ci+1
  - Ci <=lex Ci+1
- Similarly
  - Ri <=lex Ri+1

Swap last 2 colums

2. ABCDEF <= ACBDFE
becomes
2. BCEF <= CBFE
becomes
2. BE <= CF
i.e.
Second col$<=_{lex}$ Third col

Figure 10.9 p. 351

Incomplete Symmetry Breaking

0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	**A**	**B**	**C**
0	1	0	**D**	**E**	**F**
1	0	0	**G**	**H**	**I**

0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	1	**C**	**B**	**A**
0	1	0	**F**	**E**	**D**
1	0	0	**I**	**H**	**G**

0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	**C**	**B**	**A**
0	1	0	**F**	**E**	**D**
1	0	0	**I**	**H**	**G**

Section 10.4.5, p348

GAC-Lex

- We can always insist on $<=_{lex}$ in any number of dimensions
- $<=_{lex}$ is complete in some special cases
- GAC-Lex propagates the $<=_{lex}$ constraints as much as possible and in linear time
  - Frisch, Hnich, Kiziltan, Miguel, Walsh, CP 02
- Makes lexicographic ordering very attractive
  - e.g. Used in ILOG research code to solve problems with more than 10^{100} symmetries
- Other constraints researched in similar ways
  - e.g. multiset ordering, lex-all-perm

Section 10.4.3, p346

Lex-Leader with All-Different

- Suppose we have an *arbitrary* group of variable symmetries
  - And *all different* applies to the set of n variables involved
- Then we can break all symmetries with no more than n-1 constraints
  - Puget, IJCAI 05
- Why?
  - Lex-leader constraints are …
    - $\langle A,B,C,D\rangle <= \langle A,C,B,D\rangle$
  - But A=A, and B is different from C. So constraint can be simplified to
    - $B < C$
  - There are at most n(n-1)/2 constraints like this
  - And more subtle reasoning gets it to n-1

Tip

- *Do (try to) publish results that you think are obvious*

Do publish obvious results

- I knew Puget's result before Puget
  - Because a PhD student told me about it
  - S/he thought it was obvious!
  - It wasn't, it was new.
  - Puget ***rightly*** gets credit for this result
- Try to publish stuff even if you think it's obvious
  - Unless there's good evidence it is known
  - The worst that happens is you get rude reviews
    - That's happened to me!
- You often think that
  - "If I can do it, it must be simple and obvious."
- Most ideas are obvious when you think of them
  - Even if you have thought about them for years!

Symmetry Breaking in Constraint Programming

- Three main approaches to symmetry breaking
  - reformulate the problem
  - **adapt search algorithm to break symmetry**
  - add constraints before search

Section 10.5, p 350

Adapting Search Algorithms

- Two main approaches to talk about
- Symmetry Breaking During Search
  - add a constraint at each node to rule out symmetric equivalents in the **future**
- Symmetry Breaking by Dominance Detection
  - check each node before entering it, to make sure you have not been to an equivalent in the **past**
- ***Any*** implementation of either is inevitably implementing computation group theory
- We can benefit in both cases by using CGT consciously

SBDS: Symmetry Breaking During Search

- History…
- Symmetry Excluding Search Trees
  - Backofen & Will, CP 98 Workshop, CP 99
- Symmetry Breaking During Search
  - Gent & Smith, ECAI 2000
  - leads to generic name SBDS
  - as often happens, second paper more cited than first
- Implemented in Mozart (Backofen), Solver (Gent/Smith), Eclipse (Harvey/Petrie)

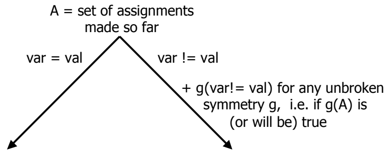
Symmetry Breaking During Search (SBDS)

- A symmetry can be eliminated if we describe its effect on the assignment of a value to a variable
- e.g. for *n*-queens, we can completely eliminate all symmetry by describing the 7 symmetries
  - we can ignore the 8^{th} symmetry, the identity

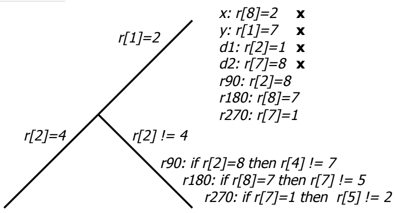
Symmetries of *n*-queens

- Variable *r[i]* represents the queen on row *i*
- The values represent the columns (1 to *n*)
  - *x(r[i]=j) → r[n-i+1]=j*
  - *y(r[i]=j) → r[i]=n-j+1*
  - *d1(r[i]=j) → r[j]=i*
  - *d2(r[i]=j) → r[n-j+1]=n-i+1*
  - *r90(r[i]=j) → r[j] = n-i+1*
  - *r180(r[i]=j) → r[n-i+1]=n-j+1*
  - *r270(r[i]=j) → r[n-j+1]=i*

Symmetry Breaking on Backtracking

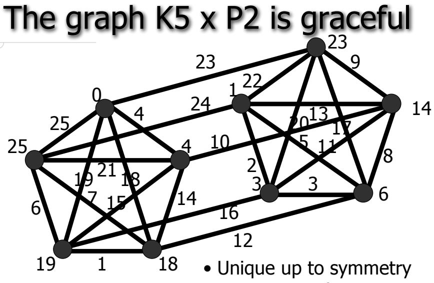


Example – 8 queens



The user need only write a function for each symmetry, describing its effect. The rest is automatic.

The graph K5 x P2 is graceful



- Unique up to symmetry
- Petrie & Smith 03
- Found using SBDS

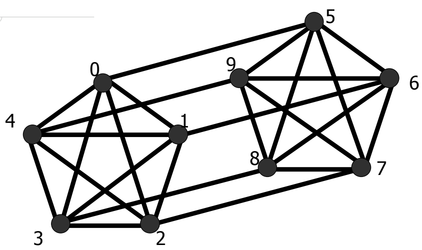
SBDS pros and cons

- Key advantages of SBDS
  - it is guaranteed to respect the heuristic
    - i.e. the *first* solution found by any heuristic is never ruled out
    - this advantage is shared by SBDD (to come later)
    - adding constraints before search can conflict with heuristics
  - we can delete symmetries when they are guaranteed broken
- Key problems with SBDS
  - needs a separate function for each symmetry
    - needs to be specified at compile time
  - many duplicate symmetry constraints
    - suppose there are n variables and m values
      - no more than nm symmetry constraints possible at the root
      - but we might add (say) n!, mainly duplicate, constraints

Computational Group Theory (CGT)

- Permutations groups
- Compact representation using stabilizer chain
- Stabilizer(G,x) = permutations of G that leave x unchanged
- Orbit(G,x) = set of elements to which x can be mapped be elements of G

The graph K5 x P2



- 240 graph symmetries = 5! x 2

Stabilizer chain

- G0 = G, Orbit(G0, 0) = {0,1,2,3,4,5,6,7,8,9}
- G1 = Stab(G0, 0), Orbit(G1,1) = {1,2,3,4}
- G2 = Stab(G1,1), Orbit(G2,2) = {2,3,4}
- G3 = Stab(G2,2), Orbit(G3,3) = {3,4}
- G4 = Stab(G3,3), Orbit(G4,4) = {4}

Generators and Groups

- Consider simple problem
  - $A^3 + B^3 + C^3 + D^3 = E^3 + F^3 + G^3$
  - A…G :: 1..50
- Given any solution, we can permute the values of ABCD and of EFG, independently.
- (A B C D) is *permutation* cycling A through D
  - (A D) swaps A and D
  - (E F G) cycles E through G
  - (E G) swaps E and G
- These four permutations *generate* the *group* of different ways to construct one solution from another.
  - there are 4!3! = 144 group elements

GAP and GAP-ECLiPSe

- GAP is world leading computational algebra package
  - accepts generators and computes using the *whole* group
  - e.g. we pass four elements (A B C D), (A D), (E F G), (E G)
    - GAP computes with full group of size 144
    - 144 is unusually *small*
    - number of generators is *at worst* logarithmic in size of group
- GAP-ECLiPSe
  - define generating elements and pass to GAP
  - constraint problem searched in ECLiPSe
  - dominance check written in GAP
  - could be done in any pair of Algebra/Constraint packages

SBDS using Computational Group Theory

- Want to retain advantages of SBDS and eliminate disadvantages
- Can easily remove need for lots of symmetry functions [e.g. McDonald 01]
  - just include generators of symmetry group
  - i.e. enough symmetries so that an arbitrary symmetry can be constructed of the generators composed enough times

SBDS using Computational Group Theory

- Now want to remove existence of duplicate symmetries …
  - again, pretty easy [McDonald 01]
- while retaining deletion of broken symmetries
  - this is not so easy [Gent, Harvey, Kelsey, 02]
  - will just sketch how to do this using CGT
- Key concepts are
  - stabiliser chain
  - right transversal

SBDS using Computational Group Theory

- stabiliser chain
  - form stabiliser of each search decision var=val
    - in context of previous stabiliser in stabiliser chain
      - starting from original group G at root of tree
- right transversal
  - orbit of this stabiliser in previous element of chain
    - no other element of group gives essentially different element
  - all possible distinct SBDS constraints formed by navigating transversals of stabiliser chain in all possible ways
- Now to avoid considering broken symmetries

SBDS using Computational Group Theory

- We no longer construct SBDS constraints
- We traverse stabiliser/transversal chain
  - ***searching*** for values that would be removed for constraints if we added them
  - elements of transversal correspond to elements of a SBDS constraint
    - we do not continue searching if an element corresponds to a broken symmetry
    - this corresponds to removing broken symmetries

SBDS using Computational Group Theory

- Further efficiency gains from lazy evaluation
- We do not continue search if we do not know the value of an element
  - unless it's the last element in structure
    - corresponding to a value that would be removed by an SBDS constraint
  - this saves huge amounts of search
  - but reduces number of values removed by SBDS constraints
- Lets us use groups up to 10^9 in BIBD's
  - compared to 10^4 using conventional SBDS

Tip

Be known for something

Tip

Be idiosyncratic

Tip

Be odd

Be idiosyncratic

- If it's something you would do anyway…
  - Juggle in talks
  - Wear odd socks
  - Wear Ben & Jerry's t-shirts
- It gives you something to talk to people about

June 2007 Ian Gent, ACP Summer Sc

Be idiosyncratic

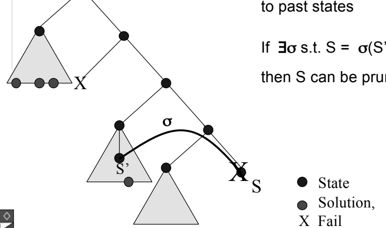
- P.s. it's ok to get known for your work too ..
  - Oh yes, Geetha, she's the one who proved P=NP
  - Chris wrote Minion
- But for mortals among us juggling is a useful backup

Section 10.5.2, p357

SBDD: Symmetry Breaking by Dominance Detection

- Symmetry Breaking by Dominance Detection
  - Fahle, Schamberger, Sellmann, 2001
  - Foccaci, Milano, 2001
  - prefigured by Brown, Finkelstein, Purdom, 1988
- Do not search a node if you have searched its equivalent before
  - *check* before entering a node

SBDD: Past states as nogoods



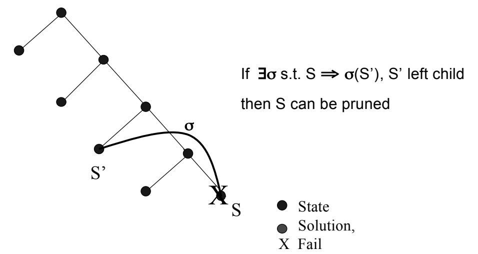
Avoids generating states isomorph to past states

If $\exists\sigma$ s.t. S = σ(S'), S' past state

then S can be pruned

Generalized nogoods

Only look at the roots of left subtrees



Dominance Check

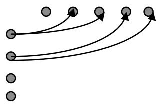
- Hand coded checking procedure
- Express check as a constraint problem
  - State inclusion
- Use sub graph isomorphism
- Use graph isomorphism
- Use CGT
  - Gent, Harvey, Kelsey, Linton wrote a *generic* dominance checker
  - works for any constraint problem
  - user has only to define the *group* acting on the CSP

Using graph theory

- Each state is represented by a graph.
- Two states are equivalent if their graphs are isomorph
- A state dominates another state if its graph is isomorph
- Symmetries of the problem translate into symmetries of the graph

Example : BIBD

0	1	1	0	0
0	0	0	1	1



- All rows and all columns can be permuted
- 0-1 variables
- Some constraints
- A partial state is shown

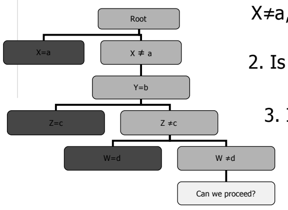
- On vertex per row, and one per column
- An edge if the variable is set to 1.

Dominance check

- Sub graph isomorphism
- Can be tackled as a separate CSP (Puget 02)
- Another approach is to store all past states
  - Dominance is then graph isomorphism (Pearson 04)

Section 10.5.5, p359

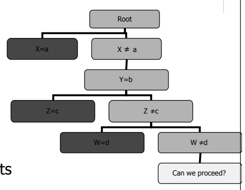
Dominance Check using CGT



1. Deduce consequences of X≠a, Y=b, Z≠c, W ≠d
2. Is X=a equivalent to anything deduced?
3. Is Y=b, Z=c equivalent to anything deduced?
4. Is Y=b, W=d equivalent to anything deduced?

Dominance Check using CGT

- *failsets*
  - sets of decisions at failed nodes
  - {X=a}, {Y=b,Z=c}, {Y=b,W=d}
- *pointset*
  - set of assigned variables at current node
  - {Y=b,X=b,V=c}
    - includes propagated values
    - assume X=b, V=c set by inference
- It is a heuristic choice to have failsets as small as possible, pointset as big as possible.



Dominance Check as Search

- Dominance check is an algebraic search
  - Aside: anyone who has written a dominance check has written a computational algebra program!
- Seek group element g and failset S s.t.
  - $S^g \subseteq$ *Pointset*
- Search process is essentially straightforward
  - but omit most algebraic details here [talk to Steve Linton for them!]
    - critical to performance by several orders of magnitude

Dominance Check as Search

- Recursive backtracking search
  - work through failset
  - find ways of mapping first element of failset to elements of pointset: try these in turn
  - recurse for second element, fixing the mapping of the first element
- GAP allows us to make computation efficient
  - work in *stabilizers,* subgroups fixing work done so far
  - use *transversals*, the places that elements can map to
  - use Schreier vectors for efficiency

Dominance Check as Search

- is there *g, S* with $S^{g} \subseteq \mathit{Pointset}$?
  - *S in failsets:* {X=a}, {Y=b,Z=c}, {Y=b,W=d}
  - *Pointset:* {Y=b,X=b,V=c}
- Use the fact that Y=b shared between second two failsets

- Can we map X=a into *Pointset?*
  - no!
- Can we map Y=b into *Pointset*?
  - Yes!
    - Y=b → Y=b
  - Use Stabilizer of Y=b
  - Can we map Z=c into {X=b,V=c}?
    - no!
  - Can we map W=d into {X=b, V=c}?
    - Yes!
    - {Y=b,W=d} → {Y=b, V=c}
- **Dominance detected so backtrack**

Inference from Dominance

- Dominance can deduce domain removals
  - Where a domain value would make dominance succeed, it can be removed
  - Already known (Fahle et al, Brown et al)
  - Again generically implemented
- Work heuristically
  - i.e. only report removals we find easily
  - Petrie [SymCon03] shows this can increase search compared to SBDS, which finds all removals

Section 10.5.3, p357

Symmetry Breaking Heuristics

- Meseguer and Torras,2001
- Break symmetry *as early as POSSIBLE*
- Use Heuristics to do this
  - Branch into parts of the search space with little symmetry
  - M&T give a heuristic to do this
- This idea has not been explored further

Tip

Put up with your collaborators being incredibly annoying

Put up with your collaborators being incredibly annoying

- Working with other smart people is the best part of the job …
  - I should know, I've worked with…
    - … Well, the same bunch of people.
- And sometimes they can be incredibly annoying!
  - If you've collaborated you know what I mean.
  - Odd, since you and I are never annoying!
    - P.s. my wife is never incredibly annoying of course!
- Be nice to them anyway
- A research group can be like a family
  - And I mean the good and bad ways
- Don't be an abused partner
  - But don't be surprised at fights the day before paper deadlines

The definition of a GE-tree

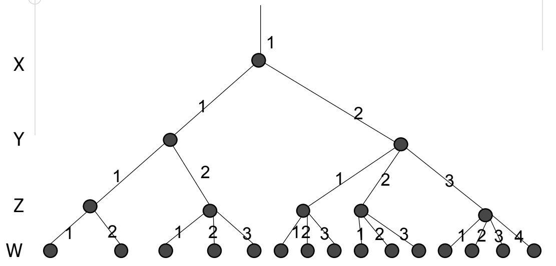
Section 10.5.6, p360

- A Group Equivalence tree (GE-tree) for a CSP with symmetry group G is any search tree T that satisfies the following two rules

1. No node of T is isomorphic under G to any other node.

2. Given a full assignment A, there is at least one leaf of T which lies in the orbit of A under G.

A GE-tree for *S_n* on values 1,2,3,4 … n



S_n on values

- There's a simple rule for this case…
- For each variable,
  - try each value used for previous variables *and one more*
- This and other cases given by
  - van Hentenryck et al 03
- Generalised by
  - Roney-Dougal et al 04

X	Y	Z	W
1	1	1	1
1	1	1	2
1	1	2	1
1	1	2	2
1	1	2	3
1	2	1	1
1	2	1	2
1	2	1	3
1	2	2	1
1	2	2	2
1	2	2	3
1	2	3	1
1	2	3	2
1	2	3	3
1	2	3	4

Constructing GE-trees for value symmetries

- Suppose that the value group of a CSP consists only of pure value symmetries.
- At each node N we proceed as follows:
  - Compute the pointwise stabiliser *G_N* of all values occuring in the path from the root to N.
  - Pick a variable *X* which has not yet been assigned.
  - Compute the orbits of *G_N* on the domain of *X.*
  - For each orbit, select a representative, and make a branch below *N* that is labelled with that representative.

Constructing GE-trees for value symmetries

- S_n on values was a special case of this
  - orbits are values assigned so far and all the rest
  - so orbit representatives are
    - all values assigned so far and one more
- Algorithm is completely general for arbitrary value symmetry groups
  - even if values move different variables differently
- Takes time $O^{\sim}((nm)^4)$ in worst case
- So …

Constructing GE-trees for value symmetries

- Value symmetry breaking is **tractable** for arbitrary value groups

S_n on variables

- GE-trees can be used on arbitrary symmetries
- There won't be such nice general results for value symmetries
- But for S_n on variables there is still a simple rule
- For each variable,
  - try each value >= value of previous variable
- Note this is the same as the naïve idea X <= Y <= Z

X	Y	Z
1	1	1
1	1	2
1	1	3
1	2	2
1	2	3
1	3	3
2	2	2
2	2	3
2	3	3
3	3	3

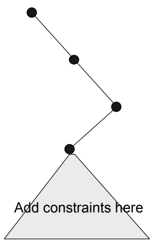
Combining methods

- When the order in which solutions are searched is the same as the one used in lex leaders, then
- SBDD, GE tree, and Lex leader can be combined together
  - E.g. see Puget, IJCAI 2005
  - Care is needed (prove it's correct!)

Section 10.5.7 p361

STAB

Add constraints before backtracking



Use the group of symmetries that leaves the current partial assignment unchanged (the stabilizer)

Can use a subset of the stabilizer

Constraints are added BEFORE backtracking

Different from SBDS/SBDD

Do not break all symmetries

Example : Matrix problem

0	1	1	0	0
0	0	0	1	1

X1	X2	X3	X4	X5
X6	X7	X8	X9	X10
X11	X12	X13	X14	X15
X16	X17	X18	X19	X20

All rows and all columns can be permuted

- Assume the first 10 variables are fixed
- One symmetry of the stabilizer is shown :
  - The first two rows are swapped
  - Columns 2 and 4 are swapped
  - Columns 3 and 5 are swapped
- Variables are ordered as shown

Example (Ct'd)

Add a lexicographic ordering constraint

0	0	0	1	1
0	1	1	0	0

X1	X2	X3	X4	X5
X6	X7	X8	X9	X10
X11	X12	X13	X14	X15
X16	X17	X18	X19	X20

[X1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20]
<=
[x6,x9,x10,x7,x8,x1,x4,x5,x2,x3,x11,x14,x15,x12,x13,x16,x19,x20,x17,x18]

Since X1..x10 fixed, it can be simplified into

[x11,x12,x13,x14,x15,x16,x17,x18,x19,x20]
<=
[x11,x14,x15,x12,x13,x16,x19,x20,x17,x18]

Sum of identical columns

Very effective improvement

0	0	2
0	2	0

X1	X2+X3	X4+X5
X6	X7+X8	X9+X10
X11	X12+X13	X14+X15
X16	X17+X18	X19+X20

[x11, x12+x13, x14+x15, x16, x17+x18, x19+x20]
<=
[x11, x14+x15, x12+x13, x16, x19+x20, x17+x18]

STAB$_{\text{Root node}}$

- At the root node, the stabilizer is the full symmetry group
- Can state constraints for a subset
- For instance, double lex for matrix problems
  - [Flener et al, CP'02]
- State that first row is smaller than all others

Section 10.9, p367

Symmetry in Other Problems

- Symmetry affects search in many other domains
  - Model checking, planning, satisfiability, theorem proving, integer programming…
- Some of the same techniques have been used/reinvented in each domain.
- It is good to be aware of what's been done
  - But I won't go into detail.

Symmetry in Satisfiability

- Lex-leader originates in a paper about SAT.
- In fact SAT is just a constraint problem in which
  - Every domain has the two values **T** and **F**
  - Every constraint has only one disallowed tuple
- Constraint propagation is ***really fast***
- So lex-leader is almost universally used
  - Work concentrates on detecting symmetry, finding good subsets of lex-leader, etc.

Symmetry in Planning

- Main work is by Fox and Long
- Similarities and differences in approach
- Symmetry detection is important
- "Almost-symmetries" are important
- Breaking approach is a bit like SBDS.

Symmetry in Model Checking

- Symmetry typically arises by use of duplicate subcircuits.
- Has not traditionally used computational algebra.
- CA has started to be used recently by Donaldson *et al*

Symmetry in Theorem Proving

- Automated theorem proving is usually undecidable
  - Though SAT is an example of a propositional case where it is NP complete
- Approach has usually been to devise *proof systems* which incorporate symmetry in some way.
- Quite a lot done in this area.

Symmetry in Mathematics

- This is almost a joke
  - Group theory is a major research area in mathematics and is **exactly** the study of symmetry
- The major *free* computational algebra system is **GAP**
- Mention a few relevant things
  - nauty (Brendan McKay) for graph isomorphism
  - GRAPE (Len Soicher) for graphs in groups in GAP
  - www.designtheory.org (Soicher) for computational resources in latin squares, designs in general, etc.
  - Partition backtrack is a key and very hard algorithm

Section 10.10, p368

Pros and Cons

- Three main approaches to symmetry breaking
  - reformulate the problem
  - adapt search algorithm to break symmetry
  - add constraints before search
- Each has advantages and disadvantages

Pros and Cons

- reformulate the problem
- Pros
  - Can lead to wonderful improvement in search
  - Can be easy to combine with other methods
- Cons
  - Can need magic
  - No general method
  - Can lead to complicated models

Pros and Cons

- adapt search algorithm to break symmetry
- Pros
  - entirely general given only group generators
  - gives unique solution from each equivalence class
  - never conflicts with heuristic
- Cons
  - complexity of dominance test can dominate
  - constraint programmers can't write generators

Pros and Cons

- add constraints before search
- Pros
  - can have extremely low overheads
    - can be good tradeoff on amount of symmetry broken
  - can deal effectively with commonly occurring symmetry groups (e.g. matrix models)
- Cons
  - doesn't eliminate all symmetric versions
  - can conflict with heuristics

Pros and Cons

- Combining Symmetry Breaking Methods
- Pros
  - Get the best of all worlds
  - Easy when combined with reformulation
  - Can be done with great care in other cases
- Cons
  - It is **MUCH** harder than you think
  - Bad things happen when you are not looking
  - It is **MUCH MUCH** harder than you think.
  - Really, it's hard. I'm not kidding. It's difficult.

Tip

- Get good at writing papers
- Well, duhhhhhh
- But what I mean is
  - Get good at all the boring stuff
  - Judging the right content for a paper
  - Writing it well
  - Thinking of a good place to send it
  - Meeting the deadline instead of not …

Tip

- Get good at ***not*** writing papers
- Sometimes ideas don't work out
  - You know it in your heart of heart
  - When you realise this, stop work
    - Even if you have put serious work into it
    - P.s. this does NOT apply to your PhD thesis
  - Ever heard of the Concorde fallacy?
- Much better to use the time to do the other thing you could be working on
  - Instead of getting bad reviews
  - OR getting a bad paper accepted
    - Which can be worse for your reputation

Tip

- Get good at ***balancing*** the last two tips!

Nearly done!

- Now for some slightly more serious tips

What should I do if ….

- I want to exploit symmetry in search
  - But I don't want to be a symmetry researcher?
- Understand a little about groups
- Understand lex-leader in principle
- Get used to the idea of adding lex-leader constraints
- Be prepared NOT to break all symmetry

What should I do If …

- I want to do a PhD in symmetry?
- Read a most excellent survey that has recently been written
  - And catch up with literature published since
- Look for the holes
  - And ones which are not too deep and technical
- Do some new research and get your PhD
- And visit St Andrews!

What can I do a Symmetry PhD on?

- Symmetry detection
  - before and during search
- Symmetry in propagation
- Combination of symmetry breaking methods
- Automatic reformulation methods
- Symmetry introduction & relaxation
  - almost symmetries
- Symmetry and implied constraints
- Constraint techniques for Group Theory
- And lots of other things.

Two great long term results of summer schools …

- If you have been, thanks for listening!
    - John Ebdon, BBC Radio