Research Frontiers - Constraints: lecture 3

Karen Petrie
karenpetrie@computing.dundee.ac.uk

Ordering Heuristics

Variable Ordering

- A part of any search algorithm is choosing a variable that has not yet been instantiated and assigning it a value from its domain.
- There are both static and dynamic variable ordering heuristics available that decide how to choose this next variable.

Variable Ordering

- One such heuristic is SDF (Smallest Domain First), which comes from the fail-first principle.
- The SDF heuristic selects from the set of unassigned variables the next variable with the fewest remaining values in its domain.
- Essentially this allows us to discover a dead end sooner than we would have and as a result reduce the overall size of our search tree.

Variable Ordering

- This heuristic becomes much more useful when dealing with a problem with noticeable variances in the size of domains.

Variable Ordering

- Another heuristic for variable ordering, often used as a tie-breaker is the degree heuristic.
- This heuristic attempts to choose the unassigned variable that is involved in the most constraints with other unassigned variables.
- This reduces the number of children of each node in the search tree by decreasing the domain sizes of other variables.

Value Ordering

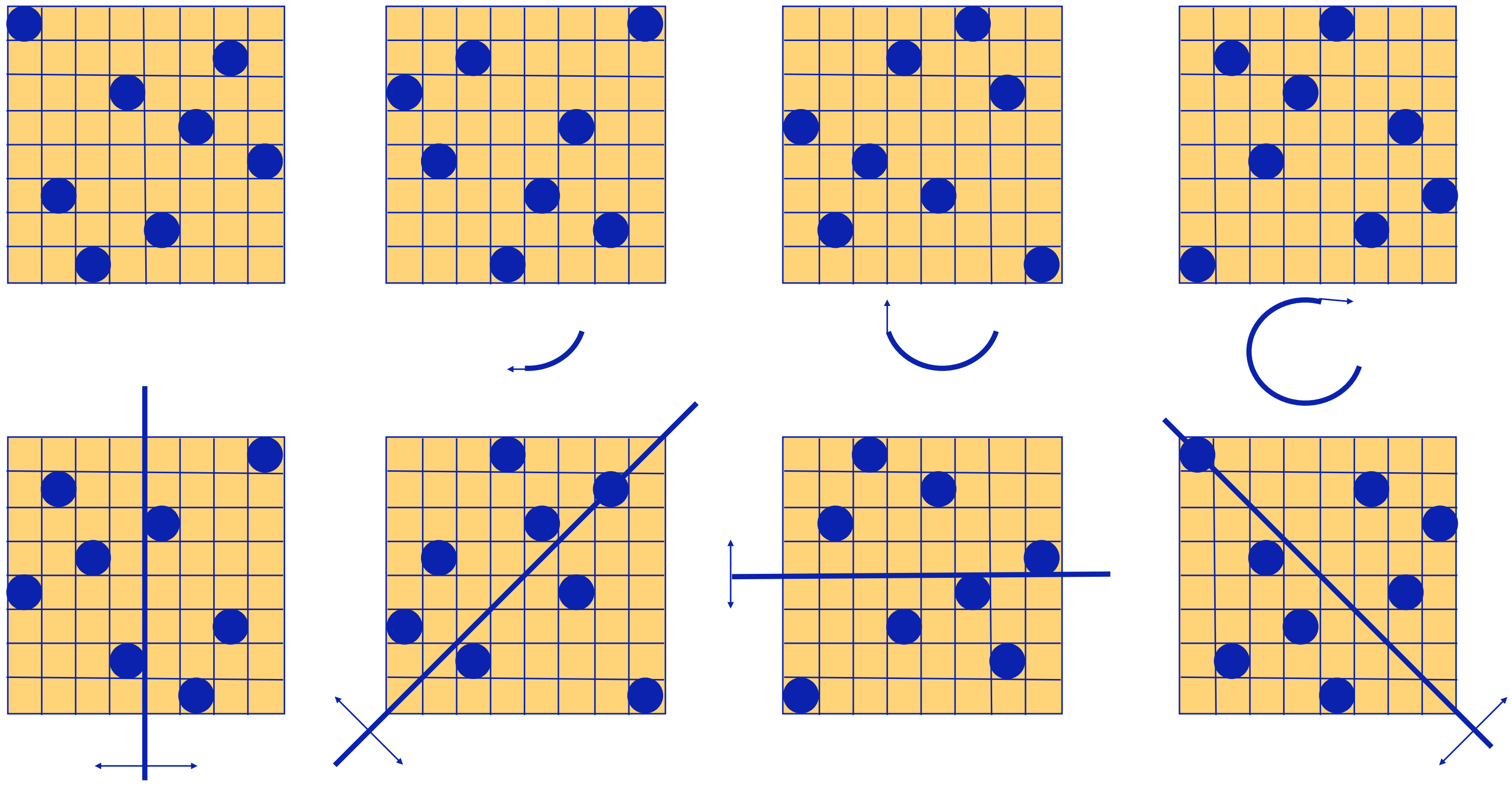
- After we have a variable, we must assign it a value.
- The way in which we choose values, or value ordering, is also important
    - because we want to branch as often as possible towards a solution
    - (though value ordering is a waste of time if we are looking for all solutions).

Value Ordering

- The most popular heuristic for choosing a value is LCV, or least-constraining value.
- The idea is to choose the value that would eliminate the fewest values in the domains of other variables and thus hopefully steer the search away from a dead end.
- By doing so, it leaves the most choices open for subsequent assignments to unassigned variables.

Symmetry

8 symmetries of the square



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
- 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 isomorphic variants

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

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

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

Reformulation Example (continued)

- 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

Symmetry Breaking in Constraint Programming

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

Adapting Search

- 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**

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 Constraints

- Probably the grandmother of symmetry breaking
- Added ad hoc since the beginning of time
- e.g.
  - X <= Y <= Z … if acts on n 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

Lex-Leader

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

Example: a 2x3 Matrix

A	B	C
D	E	F

- E.g. swap rows and first and last column
- ABCDEF <= FEDCBA
- There are 12 symmetries group

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
- Which is not scaleable
- 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

State some lex leader constraints

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

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 search 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
- Cons
  - doesn’t eliminate all symmetric versions
  - can conflict with heuristics

How to do this in practice?

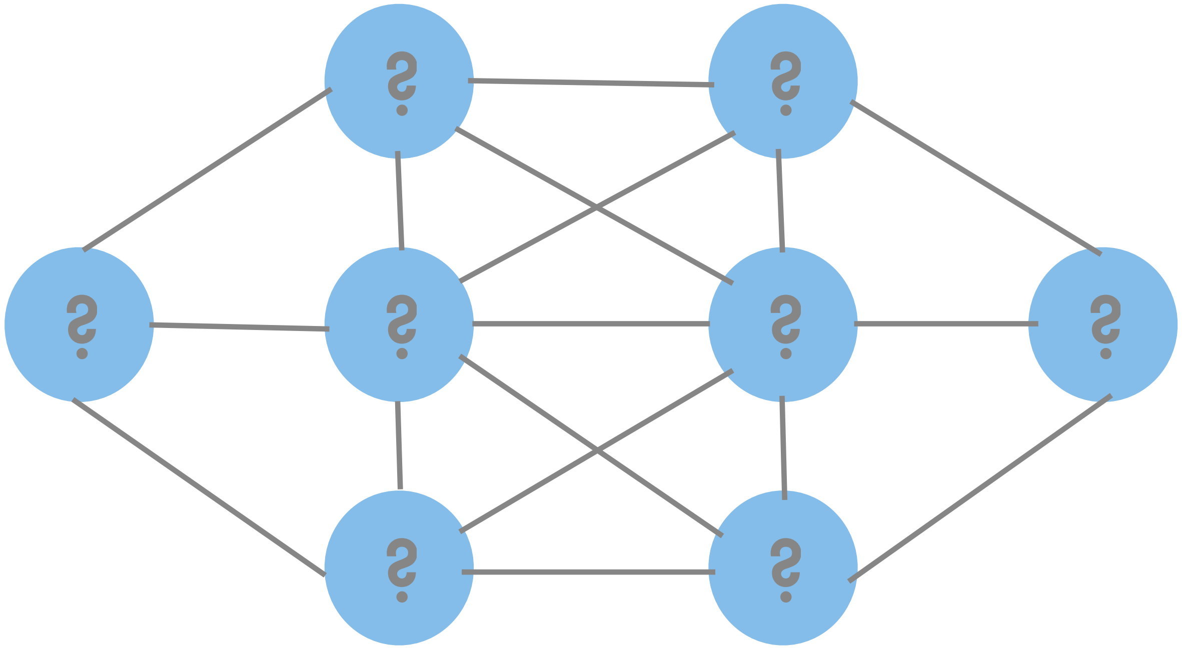
Overview of Methods

- Essence′ - High level language
- Savile Row - Converts Essence′ to Minion Input
- Minion input - lower level language
- Minion - CP solver

Future

- From now on we will be using Saville Row to create Minion
- Then editing Minion input file
- Before running file through Minion

Remember this?



Essence'

find circles: matrix indexed by [int(1..8)] of int(1..8)

such that

alldiff(circles),

| circles[1] - circles[2] | > 1,

| circles[1] - circles[3] | > 1,

| circles[1] - circles[4] | > 1,

| circles[2] - circles[3] | > 1, ……

Minion

VARIABLES

DISCRETE circles[8] {1..8}

auxiliary variables

DISCRETE aux0 {-7..7}

DISCRETE aux1 {0..7} ....

Minion ctd

SEARCH

PRINT [circles]

VARORDER [circles,
aux0,aux1,aux2,aux3,aux4,aux5,aux6,aux7,
aux8,aux9,aux10,aux11,aux12,aux13,aux14,aux15,
aux16,aux17,aux18,aux19,aux20,aux21,aux22,

aux23,aux24,aux25,aux26,aux27,aux28,aux29,

aux30,aux31, aux32,aux33]

Minion ctd

```
alldiff([circles])

weightedsumgeq([1,-1], [circles[6],circles[7]], aux32)

weightedsumleq([1,-1], [circles[6],circles[7]], aux32)

abs(aux33,aux32)

ineq(1,aux33,-1)
```

Advanced Modelling

- I will demonstrate techniques on the 8-puzzle
- You can try them out in the lab
- You should use some of these in your assignment

Using Minion Input

- we will be using Minion input at the command line
- This is more expressive than Savile Row
  - Saville Row outputs a .minion file, this is minion input
- minion <name of file>

To find all solutions

- minion -findallsols <name of file>

To see search tree

- minion -dumptree <name of file>

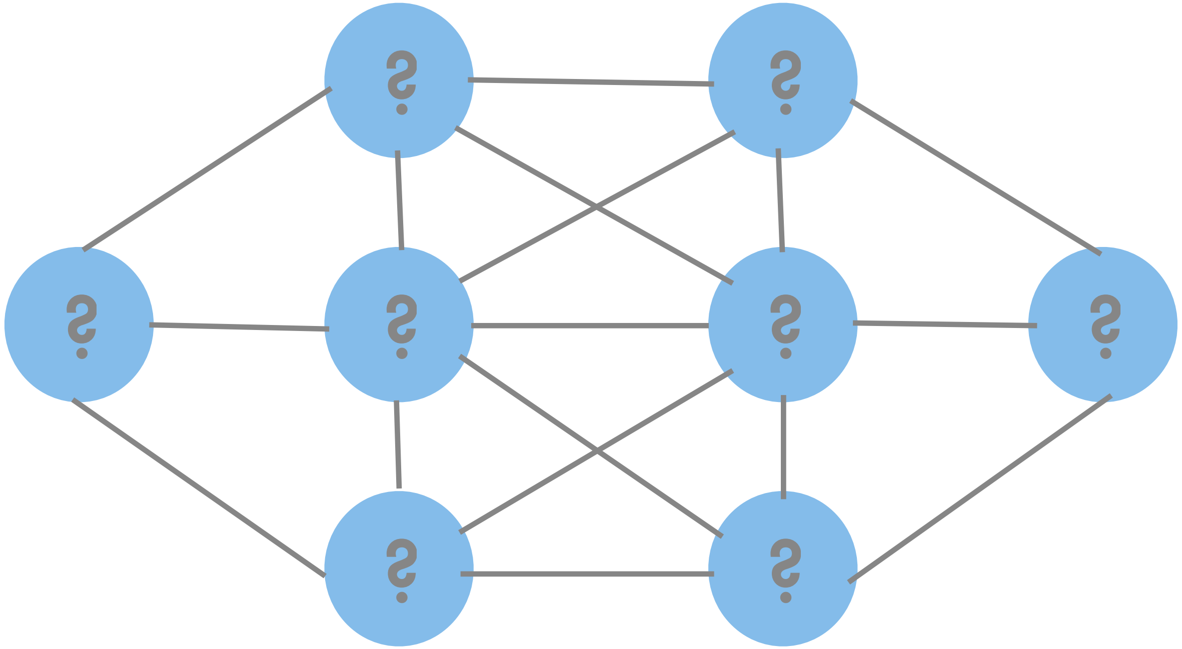
Understanding Minion

- Read the manual
  - http://minion.sourceforge.net/htmlhelp/index.html
- file called format_example.minion contains a commented example of everything.

5 Modelling Techniques

- Eliminate Symmetry - Saville Row
- Implied Constraints - Saville Row
- Reformulate Constraints - Saville Row or Minion
- Variable and Value Ordering - Minion
- Preprocessing - Minion

Implied Constraints



- The middle variables are very tightly
- constrained so they can only be 1 and 8.
- Add constraints circle[3] = 1 and circle[6]=8

Reformulate Constraints

- Comment out constraints and see if missing them decreases amount of search significantly.
- If so they could be a bottle neck, think about whether you can separate them into more constraints.

Reformulate Constraints

- Can you change the constraint to one that is the same but propagates better?
  - Look at manual for examples
- In Eight Puzzle you can:
  - alldiff to gacalldiff
  - >1 to !=1
- These may sometimes be worse!

Static Variable Ordering

- Variable ordering is given in Minion input:
- VARORDER [circles,aux0,aux1,aux2,aux3,aux4,aux5, aux6,aux7,aux8,aux9,aux10,aux11,aux12,aux13,aux14,aux15,aux16,aux17,aux18 ,aux19,aux20,aux21,aux22,aux23,aux24, aux25,aux26,aux27,aux28,aux29,aux30,aux31,aux32,aux33]

Change Var Order

- VARORDER [circles[3], circles[6], circles[1], circles[2], circles[4], circles[5], circles[6], circles[7], circles[8] aux0,aux1,...]

Value Order

- Again in Minion input, usually under variable order
- VALORDER [a,a,d,d,a,...]

Dynamic Variable Ordering

- Delete static variable and value order
- switch -varorder
  - i.e. minion -varorder sdf <name of file>

Preprocessing

- Minion has various preprocessing techniques
- switch is -preprocess
- options are, GAC, SACBounds, SAC, SSACBounds, SSAC.

Example Singleton Arc Consistency

- Assign each variable and value pair in turn
- Run GAC on these pairs, if this fails than delete these values from the domain
- Then iterate

Preprocessing

- This can take large amounts of time, but it can cut down on search drastically.
- Worth trying on difficult problems.

That is it!

- No lecture on constraints next week
- I will be at the lab tomorrow to offer any last guidance.