An Introduction to (Talks About) Constraint Programming

Ciaran McCreesh

University of Glasgow

Modelling and Solving Hard Problems

- Your typical algorithms class: in theory, some problems are (probably) exponentially hard no matter what we do.
- This talk:
  - We need to solve them anyway…
  - And we need to solve several problems simultaneously…
  - And we have awkward side constraints.

Modelling

- Express a problem as a collection of **variables**, each of which has a **domain** of possible **values**, together with a set of **constraints**.

```
% graph colouring optimisation problem, simple model

int: n;                            % number of vertices
array[1..n, 1..n] of 0..1: A;      % adjacency
array[1..n] of var 1..n: v;        % v[i] = j means vertex i has colour j

% adjacent vertices must have different colours
constraint forall(i, j in 1..n where i < j /\ A[i, j] = 1)
    (v[i] != v[j]);

% objective is to minimise chi
var 1..n: chi;
constraint chi = max(v);
solve minimize chi;
```

Solving

- SAT solvers: only 0/1 variables and CNF constraints.
- PB solvers: only 0/1 variables, linear inequalities.
- MIP solvers: only integer and 0/1 variables, linear inequalities.
- CP solvers: mixed variable types and rich constraints.
  - All different, cardinality, occurrence
  - Regular expressions on sequences
  - Array indexing
  - Lexicographic and order

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

18	23	23	245	456	456	279	378	23589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

18	23	23	245	456	456	279	378	23589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	245	456	456	279	378	23589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	245	456	456	279	378	23589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	245	456	456	279	378	23589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	245	456	456	279	378	23589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	45	456	456	79	78	589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	45	456	456	79	78	589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	45	456	456	79	78	589

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	45	456	456	79	78	89

Inference

- Each constraint has an associated algorithm, which can eliminate infeasible values from domains.
- For example, suppose we have a constraint saying that these variables must all take different values:

1	23	23	45	456	456	79	78	89

- Now each value remaining in each domain is **supported** by at least one assignment of values to each other variable. We say this constraint has achieved **generalised arc consistency**.

Propagation

- After one constraint deletes a value, this may allow other constraints to delete further values.
- We *could* keep running each constraint's algorithm in turn until we reach a fixed point. But doing this **quickly** is important.
- Is this fixed point unique?

Search

- Propagation doesn't solve the problem. So now what?
- Pick a variable, try giving it one of its values from its domain, and propagate again. If we find a solution, we're done. If we get a domain wipeout, we guessed incorrectly, so backtrack and try something else. Otherwise, recurse and try again.
- In practice, the search order is very important, and we use **heuristics**:
    - Which variable do we pick? "Smallest domain first" and "most constrained" are usually good starting points.
    - What about values?

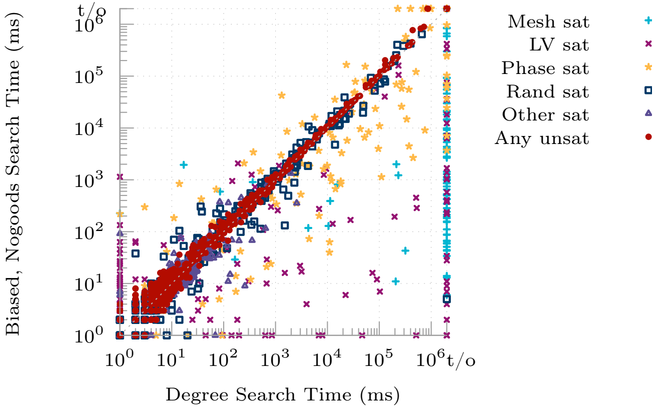
Reformulation

- We always ask: is there another model? Getting a good model matters a lot for CP.
- For that matter, CP also likes “tidied up” input.
- We can even have multiple models simultaneously, and **channel** between them.
- Even good models often exhibit **symmetries**. We can often specify additional constraints to eliminate these.

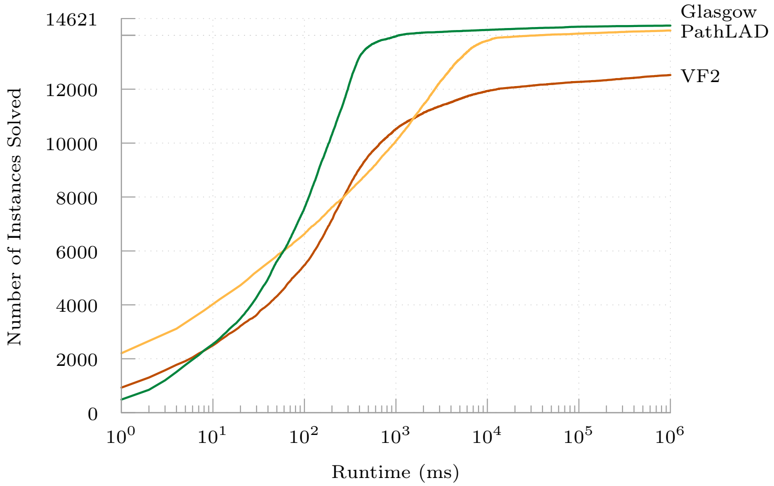
Evaluating Models and Solvers

- We have to do computational experiments.
- These are easy to do badly.
  - What instances do we use?
  - What do we compare to? Are all the solvers correct? Can we even get other people's source code?
  - Do we compare average runtimes, or something else? What if some instances time out on some solvers?
  - Are we just measuring programmer skill or programming language overheads?
  - Does our hardware behave itself?

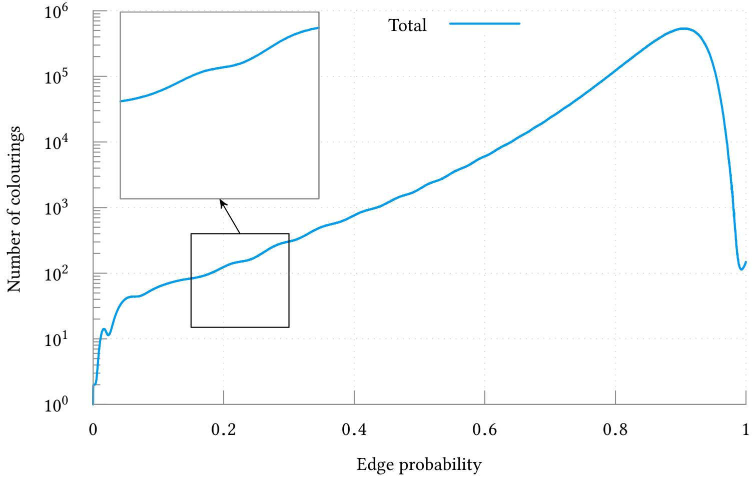
Scatter Plots



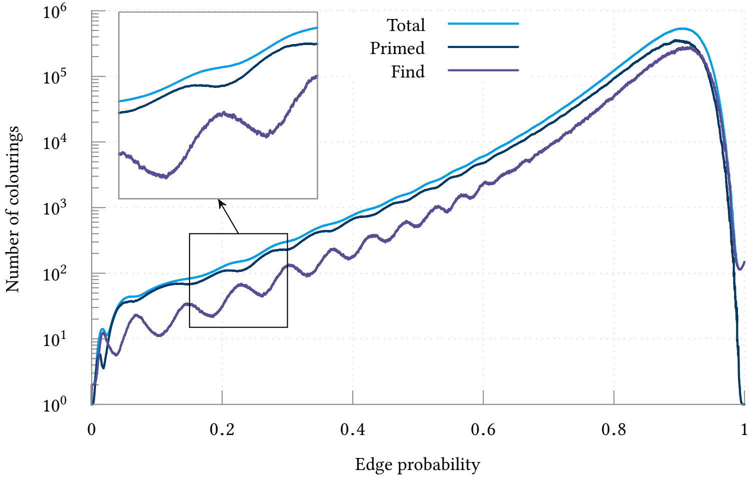
Cumulative Plots



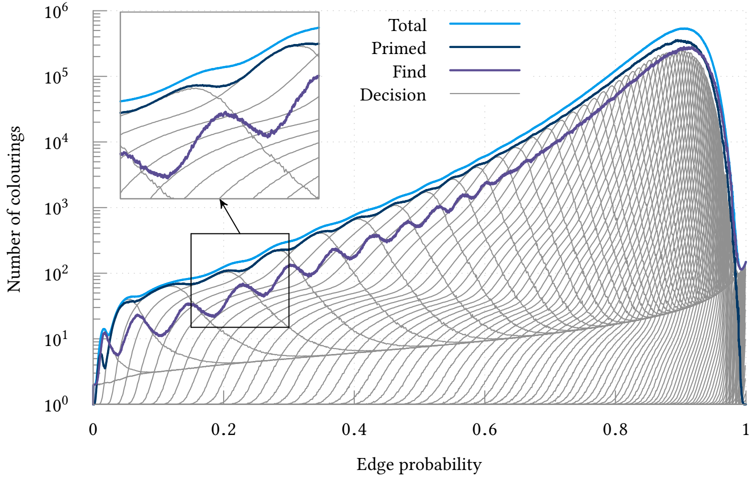
When Are Hard Problems Hard?



When Are Hard Problems Hard?



When Are Hard Problems Hard?



The Next Generation of Solvers?

- Conventional CP solvers can only reason about one constraint at a time. Future solvers may be able to do better:
  - **Learning**, by creating new constraints by analysing conflicts.
  - **Decision diagrams** have a different notion of consistency involving paths through a search tree, which can sometimes be stronger.
  - **Views** can avoid the introduction of auxiliary variables.
  - **Hybrid** solvers can solve subproblems using different solving technologies.
- High level types, such as partitions and graphs, allow for automatic reformulation.
- Better search? And what about parallelism?
- Can we do better with bad models? And can we automatically clean up bad inputs?

Should We Trust Solvers?

- How do we know solvers don't contain bugs?

https://ciaranm.github.io/

ciaran.mccreesh@glasgow.ac.uk