Modern Constraint Programming For People Who Know SAT

Ciaran McCreesh

The Classical View

- Variables, each of which has a finite domain.
- Constraints, each of which has a scope (the variables over which it is defined), and a set of feasible tuples over its scope.
  - Constraints form a network!
- Give each variable a value from its domain, such that each constraint is satisfied.

Modern Constraint Programming

- We have finite variables, integer variables, array variables, set variables, graph variables, partition variables, …
- Constraints are things like

$$x[i] + y[j] = z$$
$$\mathit{alldifferent}(a, b, c, d)$$
$$\mathit{knapsack}(w, p, n, W, P)$$

- Some kind of objective, like lexicographic optimisation, finding a diverse set of solutions, or finding a Pareto front.
  - Optimisation and satisfiable instances very common.
- Search hints.

Terminology

When I say…	I mean…
Problem	High-level description of what you're trying to do, with parameters
Model	A description of how we encode a problem
Instance	Applying a model to a set of parameter values (e.g. to get CNF)
Solution	A model
Propagation	Propagation, but somehow more complicated
Variable heuristic	Something like VSIDS
Value heuristic	Something like polarity
Nogood	Negation of a learned clause

Modelling

- Various modelling languages (Essence, MiniZinc, XCSP).
- Fairly large gap between what the user specifies and what solvers get.
- But often specific solvers are used directly as libraries.

Like This...

```
int: flour;  %no. grams of flour available
int: banana; %no. of bananas available
int: sugar;  %no. grams of sugar available
int: butter; %no. grams of butter available
int: cocoa;  %no. grams of cocoa available
constraint assert(flour >= 0, "flour should be non-negative");
constraint assert(banana >= 0, "banana should be non-negative");
constraint assert(sugar >= 0, "sugar should be non-negative");
constraint assert(butter >= 0, "butter should be non-negative");
constraint assert(cocoa >= 0, "cocoa should be non-negative");

var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes

constraint 250*b + 200*c <= flour;
constraint 2*b  <= banana;
constraint 75*b + 150*c <= sugar;
constraint 100*b + 150*c <= butter;
constraint 75*c <= cocoa;

solve maximize 400*b + 450*c; % profit
output ["no. of banana cakes = \(b)\n",
        "no. of chocolate cakes = \(c)\n"];
```

…Or This…

```
given buckets : int
letting Buckets be domain int(1..buckets)
given initialState : function int --> int
given finalState : function int --> int
given capacity : function int --> int
letting maxCapacity be max([capacity(i) | i : Buckets])
given HORIZON : int

find actions : sequence (maxSize HORIZON) of (Buckets, Buckets, int(1..maxCapacity))
find states  : sequence (maxSize HORIZON) of function (total) Buckets --> int(0..maxCapacity)

$ action i transforms state i to state i+1
such that |actions| = |states| - 1
$ cannot pour from the a bucket to itself
such that forAll (i, (bFrom, bTo, amount)) in actions . bFrom != bTo
$ can only pour if there is enough water
such that forAll (i, (bFrom, bTo, amount)) in actions . amount <= states(i)(bFrom)
$ bucket capacity
such that forAll (i, f) in states . forAll b : Buckets . f(b) <= capacity(b)
$ preservation of water
such that forAll (i, (bFrom, bTo, amount)) in actions . states(i)(bFrom) - amount = states(i+
such that forAll (i, (bFrom, bTo, amount)) in actions . states(i)(bTo) + amount = states(i+1)
such that forAll (i, (bFrom, bTo, amount)) in actions .  forAll b : Buckets . !(b in {bFrom,
$ after an action, either the source bucket is empty or the target bucket is full
such that forAll (i, (bFrom, bTo, amount)) in actions . states(i+1)(bFrom) = 0 \/ states(i+1)
$ initial and final state
such that states(1) = initialState such that states(|states|) = finalState
$ minimise the number of actions
find nbActions : int(0..HORIZON)
such that nbActions = |actions|
minimising nbActions
```

…Or Maybe This…

```
<instance format="XCSP3" type="CSP">
  <variables>
    <array id="x" size="[5]" note="x[i] is the ith value of the series
      0..4
    </array>
    <array id="y" size="[4]" note="y[i] is the distance between x[i] a
      1..4
    </array>
  </variables>
  <constraints>
    <allDifferent> x[] </allDifferent>
    <allDifferent> y[] </allDifferent>
    <group class="channeling">
      <intension> eq(%0,dist(%1,%2)) </intension>
      <args> y[0] x[0] x[1] </args>
      <args> y[1] x[1] x[2] </args>
      <args> y[2] x[2] x[3] </args>
      <args> y[3] x[3] x[4] </args>
    </group>
  </constraints>
</instance>
```

…Or Possibly This…

```
class Money : public Script {
protected:
  static const int nl = 8;
  IntVarArray le;
public:
  Money(const Options& opt) : Script(opt), le(*this,nl,0,9) {
    IntVar s(le[0]), e(le[1]), n(le[2]), d(le[3]), m(le[4]), o(le[5]), r(le[6]), y(le[7]);

    rel(*this, s, IRT_NQ, 0);
    rel(*this, m, IRT_NQ, 0);
    distinct(*this, le, opt.ipl());

    IntVar c0(*this,0,1), c1(*this,0,1), c2(*this,0,1), c3(*this,0,1);
    rel(*this,    d+e == y+10*c0, opt.ipl());
    rel(*this, c0+n+r == e+10*c1, opt.ipl());
    rel(*this, c1+e+o == n+10*c2, opt.ipl());
    rel(*this, c2+s+m == o+10*c3, opt.ipl());
    rel(*this, c3     == m,       opt.ipl());

    branch(*this, le, INT_VAR_SIZE_MIN(), INT_VAL_MIN());
  }

  virtual void print(std::ostream& os) const {
    os << "\t" << le << std::endl;
  }

  Money(Money& s) : Script(s) { le.update(*this, s.le); }
  virtual Space* copy(void) { return new Money(*this); }
};
```

. . . Or Apparently The Cool Kids Use Python Now?

```
import numpy as np
from cpmpy import *

e = 0 # value for empty cells
given = np.array([
        [e, e, e,  2, e, 5,  e, e, e],
        [e, 9, e,  e, e, e,  7, 3, e],
        [e, e, 2,  e, e, 9,  e, 6, e],
        [2, e, e,  e, e, e,  4, e, 9],
        [e, e, e,  e, 7, e,  e, e, e],
        [6, e, 9,  e, e, e,  e, e, 1],
        [e, 8, e,  4, e, e,  1, e, e],
        [e, 6, 3,  e, e, e,  e, 8, e],
        [e, e, e,  6, e, 8,  e, e, e]])

puzzle = intvar(1,9, shape=given.shape, name="puzzle")

model = Model(
        [AllDifferent(row) for row in puzzle],
        [AllDifferent(col) for col in puzzle.T])

for i in range(0,9, 3):
    for j in range(0,9, 3):
            model += AllDifferent(puzzle[i:i+3, j:j+3])

model += (puzzle[given!=e] == given[given!=e])

if model.solve():
    print(puzzle.value())
else:
    print("No solution found")
```

Reformulation

- We care a lot about finding a good model for a problem.
    - Or possibly multiple good models, with channeling constraints.
- Changing the model is totally legitimate and encouraged.
- Some of this is automated by some modelling languages.

When is CP Actually Good?

- When mixing several kinds of constraints, where they interact in interesting ways.
- If it's impractical to encode to something lower level.
- When you want a human-understandable model.
- Optimisation with hard constraints.

Variables

- Multiple representations are common:
  - Small domains: bitsets.
  - Large domains: just store lower and upper bounds, or possibly a set of intervals.
  - Lots of Boolean variables?
- Structured variables *might* have their own representations.
- No agreement on copying vs trailing.

Constraints

- A constraint defines one or more *inference algorithms* or *propagators*.
- At a minimum, these have to be *checking*, but ideally they do more.
  - Traditionally, delete unsupported values from domains.
  - Ideally, also give information on when they might infer something.
  - In newer solvers, give precise explanations for deletions.
  - In future solvers, maybe even give likelihoods.
  - Often stateful algorithms.
- Extensional representations are possible, but used sparingly.

Consistency

- Arc consistency, for constraints between two variables A and B:
  - For each value v in A, there is a value w in B such that the constraint is satisfied, and vice-versa.
- Generalised arc consistency (GAC), or domain consistency: every value in every variable exists in at least one solution to the constraint.
- Bounds consistency: the upper and lower bounds of every variable exist in at least one solution to the constraint.

Consistency Can Be Hard

$$2x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 = 5$$

- GAC is NP-hard.
- Solvers might do it anyway for small numbers.

Consistency Can Be Hard

$$2x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 = 5$$

- GAC is NP-hard.
- Solvers might do it anyway for small numbers.

$$2x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 \geq 5$$
$$2x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 \leq 5$$

- GAC on two inequalities is easy.
- The same as bounds consistency.
- Doesn't give bounds consistency on the equality, though.

Do We Care About Consistency?

- Possibly due to all-different, some people see GAC as being the “right” thing for constraints to do.
  - It’s the best thing we can do if we can only use one constraint at a time.
- Solver authors tend to care more about “what can we do fast?”.
  - Which potentially includes solving knapsack…
- GAC is rarely the *best* thing to do, but is often a quite good starting point.

Propagation

- Constraints can propagate more than once!
- Run every constraint until we reach a fixed point.
- Is this fixed point unique?
- Lots of research and engineering: doing this efficiently.
  - Events and clever queues…
  - Watches…
  - Determining a static ordering…

Constraint Propagation is Slow

- Sometimes we run lots of expensive propagators many times.
- A full round of propagation can take seconds.

Constraint Propagation is Fast

- Can infer thousands or millions of facts per clock cycle.
- Even if a SAT encoding achieves the same level of consistency as a CP solver, sometimes the CP solver gets there much faster.
  - Better constant factors from propagators.
  - CNF encoding size can kill SAT solver performance, even if it's only a linear factor difference.

Search

- Backtracking search, propagating at each node.
- Specify *variable* and *value* ordering heuristics, and decision variables.
- Smallest domain first, most constrained.
- Re-weight constraints over time.
- If a modeller is good:
    - Programmed search.
    - Branching on expressions.
    - Constraint-based local search.

Restarts

- Much less frequent than in SAT.
- Mostly used to boost weighted variable-ordering heuristics.
- Can handle nogoods from restarts efficiently.

Lazy Clause Generation

- Traditional propagation: based upon guessed assignments G we know that $X \neq 5$.
- Lazy clause generation: create $\wedge G \longrightarrow X \neq 5$ as a new constraint.
- Constraints can be much more specific about failures.
- In some ways: like having a really strong CNF encoding, but creating it dynamically.

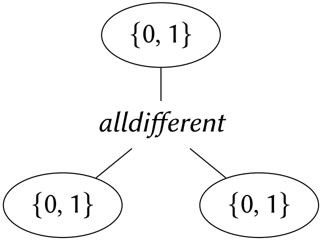
Two-Colouring a Triangle

$x_1 \in \{0, 1\}$

$x_2 \in \{0, 1\}$

$x_3 \in \{0, 1\}$

$\mathit{alldifferent}(x_1, x_2, x_3)$



Decomposing “All Different”

$x_1 \in \{0, 1\}$

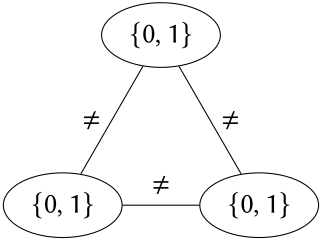
$x_2 \in \{0, 1\}$

$x_3 \in \{0, 1\}$

$x_1 \neq x_2$

$x_1 \neq x_3$

$x_2 \neq x_3$



What Does Propagation Do?

- Let's consider the constraint $x_1 \neq x_2$.
- Remember arc consistency: for each value, check whether it is supported by another value.
  - If $x_1 = 0$, we can give $x_2 = 1$, so that's OK.
  - If $x_1 = 1$, we can give $x_2 = 0$, so that's OK.
  - If $x_2 = 0$, we can give $x_1 = 1$, so that's OK.
  - If $x_2 = 1$, we can give $x_1 = 0$, so that's OK.
- Let's consider the constraint $x_1 \neq x_3$.
  - etc
- Let's consider the constraint $x_2 \neq x_3$.
  - etc
- So no values are deleted, and everything looks OK.
- Actually, there's a more efficient algorithm: $\neq$ won't do anything unless one of the variables only has one value. Some solvers won't trigger the constraint unless this happens.

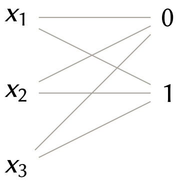
What Would a Human Do?

"Duh, obviously there's no solution! There aren't enough numbers to go around."

- Unfortunately "stare at it for a few seconds then write down the answer" is not an algorithm.
- But if we don't decompose the constraint, we *can* come up with a propagator which can tell that there's no solution.

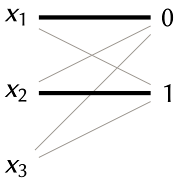
Matchings

- Draw a vertex on the left for each variable, and a vertex on the right for each value.
- Draw edges from each variable to each of its values.
- A *maximum cardinality matching* is where you pick as many edges as possible, but each vertex can only be used at most once.
- We can find this in polynomial time.
- There is a matching which covers each variable if and only if the constraint can be satisfied.
- In fact, there is a one to one correspondence between perfect matchings and solutions to the constraint.



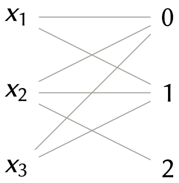
Matchings

- Draw a vertex on the left for each variable, and a vertex on the right for each value.
- Draw edges from each variable to each of its values.
- A *maximum cardinality matching* is where you pick as many edges as possible, but each vertex can only be used at most once.
- We can find this in polynomial time.
- There is a matching which covers each variable if and only if the constraint can be satisfied.
- In fact, there is a one to one correspondence between perfect matchings and solutions to the constraint.



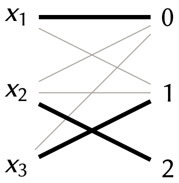
Matchings

- Draw a vertex on the left for each variable, and a vertex on the right for each value.
- Draw edges from each variable to each of its values.
- A *maximum cardinality matching* is where you pick as many edges as possible, but each vertex can only be used at most once.
- We can find this in polynomial time.
- There is a matching which covers each variable if and only if the constraint can be satisfied.
- In fact, there is a one to one correspondence between perfect matchings and solutions to the constraint.



Matchings

- Draw a vertex on the left for each variable, and a vertex on the right for each value.
- Draw edges from each variable to each of its values.
- A *maximum cardinality matching* is where you pick as many edges as possible, but each vertex can only be used at most once.
- We can find this in polynomial time.
- There is a matching which covers each variable if and only if the constraint can be satisfied.
- In fact, there is a one to one correspondence between perfect matchings and solutions to the constraint.



How do Humans Solve Sudoku?

18	23	23	245	456	456	279	378	23589

How do Humans Solve Sudoku?

18	23	23	245	456	456	279	378	23589

How do Humans Solve Sudoku?

1	23	23	245	456	456	279	378	23589

How do Humans Solve Sudoku?

1	23	23	245	456	456	279	378	23589

How do Humans Solve Sudoku?

1	23	23	245	456	456	279	378	23589

How do Humans Solve Sudoku?

1	23	23	245	456	456	279	378	23589

How do Humans Solve Sudoku?

1	23	23	45	456	456	79	78	589

How do Humans Solve Sudoku?

1	23	23	45	456	456	79	78	589

How do Humans Solve Sudoku?

1	23	23	45	456	456	79	78	589

How do Humans Solve Sudoku?

1	23	23	45	456	456	79	78	89

Hall Sets

- A *Hall set* of size *n* is a set of *n* variables from an "all different" constraint, whose domains have *n* values between them.
- If we can find a Hall set, we can safely remove these values from the domains of every other variable involved in the constraint.
- Hall's Marriage Theorem: doing this is equivalent to deleting every edge from the matching graph which cannot appear in any perfect matching.
- So, if we delete every Hall set, we delete every value that cannot appear in at least one way of satisfying the constraint. In other words, we obtain GAC.

Finding Hall Sets?

- There are 2^n potential Hall sets, so considering them all is probably a bad idea…
- Similarly, enumerating every perfect matching is #P-hard.
- However, there is a polynomial algorithm!

Finding Hall Sets?

18	23	23	245	456	456	279	378	23589

Finding Hall Sets?

18	23	23	245	456	456	279	378	23589

$row[0]$

$row[1]$

$row[2]$

$row[3]$

$row[4]$

$row[5]$

$row[6]$

$row[7]$

$row[8]$

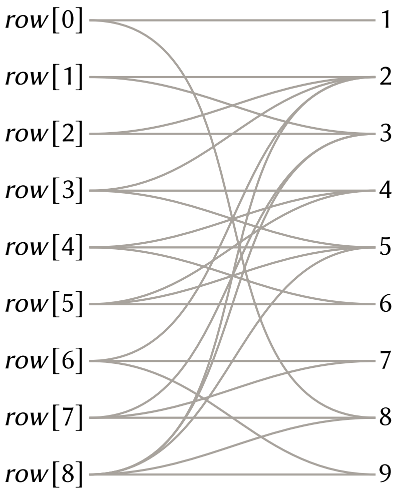
Finding Hall Sets?

18	23	23	245	456	456	279	378	23589

$row[0]$	1
$row[1]$	2
$row[2]$	3
$row[3]$	4
$row[4]$	5
$row[5]$	6
$row[6]$	7
$row[7]$	8
$row[8]$	9

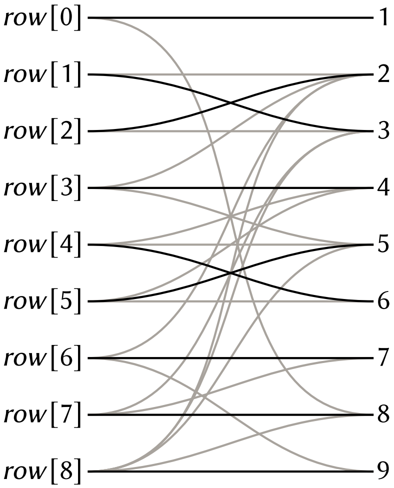
Finding Hall Sets?

18	23	23	245	456	456	279	378	23589



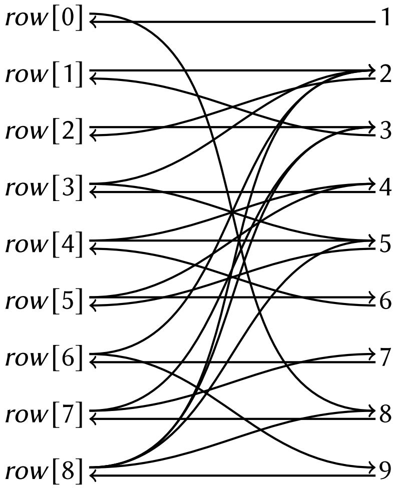
Finding Hall Sets?

18	23	23	245	456	456	279	378	23589



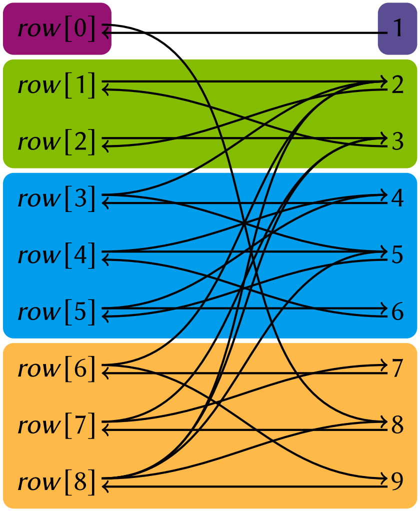
Finding Hall Sets?

18	23	23	245	456	456	279	378	23589



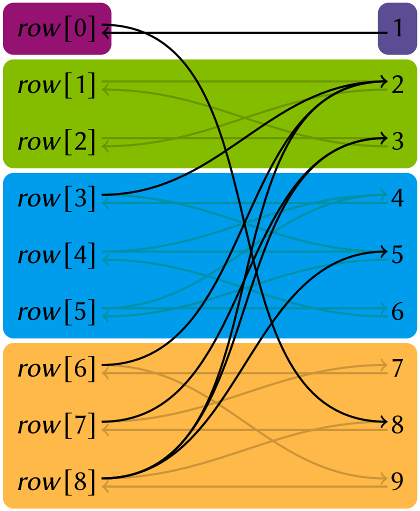
Finding Hall Sets?

18	23	23	245	456	456	279	378	23589



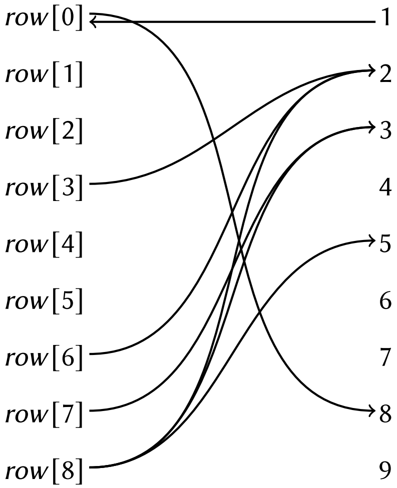
Finding Hall Sets?

18	23	23	245	456	456	279	378	23589



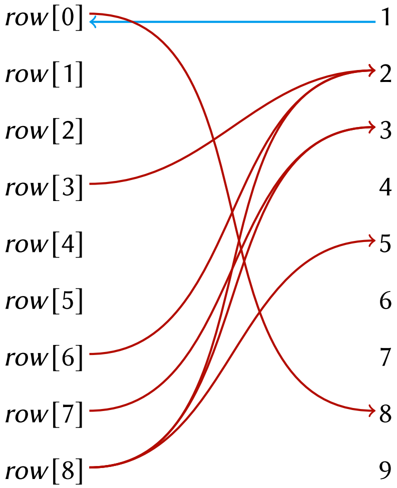
Finding Hall Sets?

18	23	23	245	456	456	279	378	23589



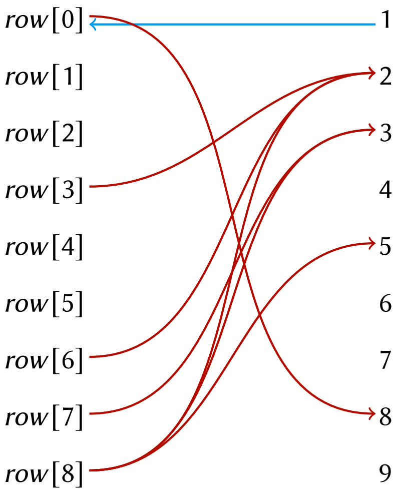
Finding Hall Sets?

18	23	23	245	456	456	279	378	23589



Finding Hall Sets?

18	23	23	245	456	456	279	378	23589



Does GAC Matter?

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

//
// Glasgow Herald 22nd Dec 2006
// easy
//
```

Does GAC Matter?

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.037s
Resolution time : 0.013s
Nodes: 1 (75.1 n/s)
Backtracks: 0
Fails: 0
Restarts: 0
```

Does GAC Matter?

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.037s
Resolution time : 0.013s
Nodes: 1 (75.1 n/s)
Backtracks: 0
Fails: 0
Restarts: 0
```

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.041s
Resolution time : 0.017s
Nodes: 1 (60.4 n/s)
Backtracks: 0
Fails: 0
Restarts: 0
```

Does GAC Matter?

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

//
// Glasgow Herald 22nd Dec 2006
// hard
//
```

Does GAC Matter?

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.041s
Resolution time : 0.024s
Nodes: 34 (1,414.5 n/s)
Backtracks: 59
Fails: 32
Restarts: 0
```

Does GAC Matter?

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.041s
Resolution time : 0.024s
Nodes: 34 (1,414.5 n/s)
Backtracks: 59
Fails: 32
Restarts: 0
```

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.039s
Resolution time : 0.022s
Nodes: 2 (90.2 n/s)
Backtracks: 1
Fails: 1
Restarts: 0
```

Does GAC Matter?

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

//
// Times 7/1/2007
// Superior (worse than ‘‘fiendish’’)
//
```

Does GAC Matter?

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.038s
Resolution time : 0.020s
Nodes: 14 (712.0 n/s)
Backtracks: 21
Fails: 11
Restarts: 0
```

Does GAC Matter?

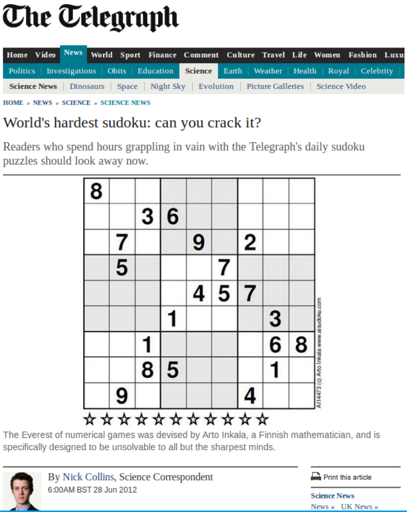
```
9 8 3 1 5 2 7 6 4
4 7 5 8 3 6 1 9 2
6 2 1 9 4 7 8 3 5
3 5 4 6 8 1 9 2 7
2 6 7 5 9 3 4 1 8
1 9 8 2 7 4 6 5 3
7 3 6 4 1 5 2 8 9
8 4 2 3 6 9 5 7 1
5 1 9 7 2 8 3 4 6

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.038s
Resolution time : 0.020s
Nodes: 14 (712.0 n/s)
Backtracks: 21
Fails: 11
Restarts: 0
```

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.040s
Resolution time : 0.022s
Nodes: 2 (91.8 n/s)
Backtracks: 0
Fails: 0
Restarts: 0
```

Does GAC Matter?



The Telegraph

Home Video News World Sport Finance Comment Culture Travel Life Women Fashion Luxu
Politics | Investigations | Obits | Education | Science | Earth | Weather | Health | Royal | Celebrity |
Science News | Dinosaurs | Space | Night Sky | Evolution | Picture Galleries | Science Video
HOME » NEWS » SCIENCE » SCIENCE NEWS

World's hardest sudoku: can you crack it?

Readers who spend hours grappling in vain with the Telegraph's daily sudoku puzzles should look away now.

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

AI14473 (c) Arto Inkala www.aisudoku.com

☆☆☆☆☆☆☆☆☆☆☆

The Everest of numerical games was devised by Arto Inkala, a Finnish mathematician, and is specifically designed to be unsolvable to all but the sharpest minds.

By Nick Collins, Science Correspondent
6:00AM BST 28 Jun 2012

Does GAC Matter?

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.037s
Resolution time : 0.057s
Nodes: 855 (14,994.6 n/s)
Backtracks: 1,687
Fails: 847
Restarts: 0
```

Does GAC Matter?

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.037s
Resolution time : 0.057s
Nodes: 855 (14,994.6 n/s)
Backtracks: 1,687
Fails: 847
Restarts: 0
```

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

1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.041s
Resolution time : 0.049s
Nodes: 83 (1,696.5 n/s)
Backtracks: 151
Fails: 80
Restarts: 0
```

Does GAC Matter?

```
?  ?  15 ?  ?  34 28 18 21 ?  ?  ?  27 ?  36 29 5  14 4  ?  ?  ?  ?  ?  ?  23 ?  ?  24 10 26 32 ?  12 19 ?
23 ?  6  ?  2  ?  ?  20 32 ?  ?  33 ?  ?  ?  ?  26 13 22 9  5  21 ?  8  17 29 ?  ?  36 1  15 28 7  35 31 ?
?  26 ?  ?  14 ?  27 36 ?  7  ?  ?  ?  ?  ?  ?  6  28 ?  ?  34 ?  23 ?  3  ?  ?  ?  30 ?  16 ?  18 9  8  29
?  32 9  28 ?  16 ?  ?  11 ?  24 ?  ?  33 ?  ?  ?  21 ?  ?  19 10 14 ?  ?  8  2  ?  12 7  ?  27 17 ?  25 ?
?  4  31 17 ?  8  23 35 10 16 6  ?  ?  ?  1  ?  30 ?  29 26 ?  ?  ?  ?  9  ?  33 ?  14 ?  ?  ?  13 36 34 ?
5  ?  33 ?  ?  ?  ?  ?  13 2  ?  ?  ?  8  32 ?  ?  ?  15 3  ?  7  12 36 34 27 ?  19 ?  26 ?  ?  23 ?  10 14
20 ?  ?  8  23 26 ?  21 ?  1  19 ?  ?  32 16 2  ?  ?  ?  12 22 ?  ?  28 ?  18 30 10 11 ?  6  ?  15 ?  ?  ?
35 11 ?  ?  13 25 36 4  ?  15 ?  ?  28 ?  6  ?  ?  1  31 23 26 ?  3  18 ?  ?  9  29 ?  ?  ?  17 33 ?  2  30
?  7  ?  4  ?  9  5  ?  17 ?  11 24 31 ?  ?  30 ?  ?  ?  35 8  29 19 32 23 ?  34 20 6  ?  ?  ?  ?  1  ?  ?
18 ?  ?  ?  22 33 29 27 ?  26 ?  2  ?  9  ?  11 14 ?  ?  ?  ?  17 16 ?  35 19 ?  ?  1  ?  ?  12 ?  ?  ?  31
3  16 ?  36 32 ?  ?  6  30 10 8  34 5  18 27 22 17 23 ?  ?  7  ?  ?  1  ?  ?  4  ?  ?  28 ?  ?  29 24 ?  ?
21 ?  ?  31 ?  ?  33 ?  ?  14 ?  ?  ?  ?  29 26 ?  4  ?  2  36 ?  ?  ?  32 ?  ?  27 ?  5  13 19 ?  18 ?  ?
34 17 ?  ?  9  ?  2  ?  ?  20 13 10 ?  ?  ?  18 7  ?  ?  ?  ?  ?  ?  22 6  ?  32 ?  15 ?  19 ?  ?  ?  4  ?
24 ?  ?  7  25 15 1  23 3  ?  31 ?  26 ?  19 14 ?  2  21 34 6  ?  ?  ?  20 ?  ?  8  17 30 33 22 36 ?  ?  10
11 27 ?  33 6  1  ?  34 25 22 ?  8  36 ?  9  ?  23 24 ?  19 31 3  ?  26 21 ?  ?  ?  4  ?  29 18 30 15 ?  ?
?  ?  ?  14 ?  ?  ?  32 ?  ?  4  26 ?  15 ?  ?  ?  20 9  ?  ?  ?  29 30 ?  ?  16 ?  22 ?  ?  7  ?  ?  ?  ?
32 ?  18 ?  ?  ?  ?  5  ?  ?  36 ?  ?  ?  35 21 ?  12 ?  16 ?  20 ?  27 ?  ?  3  ?  9  ?  1  26 8  2  ?  ?
36 ?  ?  ?  16 ?  7  ?  18 ?  ?  21 ?  ?  22 ?  ?  17 35 14 ?  24 25 ?  29 2  ?  ?  ?  ?  31 ?  3  ?  6  9
30 29 ?  12 ?  5  ?  ?  ?  ?  20 17 ?  11 21 ?  35 9  28 ?  ?  18 ?  ?  1  ?  ?  34 ?  23 ?  36 ?  ?  ?  27
?  ?  11 13 3  27 ?  16 ?  24 ?  ?  17 ?  7  ?  33 ?  5  ?  35 6  ?  ?  ?  10 ?  ?  25 ?  ?  ?  ?  30 ?  21
?  ?  ?  ?  28 ?  ?  13 ?  5  ?  ?  10 1  ?  ?  ?  31 11 ?  ?  ?  22 ?  19 21 ?  ?  2  ?  ?  ?  9  ?  ?  ?
?  ?  17 25 20 23 ?  30 ?  ?  ?  1  13 ?  24 16 29 ?  7  21 ?  27 ?  10 14 ?  35 9  28 ?  11 4  2  ?  26 12
9  ?  ?  34 18 2  25 12 14 ?  ?  35 ?  ?  ?  6  19 27 8  ?  1  16 ?  3  ?  4  ?  7  31 11 17 20 24 ?  ?  22
?  24 ?  ?  ?  35 ?  15 ?  6  ?  11 12 ?  ?  ?  ?  ?  ?  25 20 ?  ?  ?  33 36 29 ?  ?  27 ?  23 ?  ?  7  28
?  ?  14 ?  5  18 24 ?  34 ?  ?  7  ?  ?  ?  23 12 ?  10 ?  11 8  ?  ?  ?  ?  6  ?  ?  4  ?  ?  32 ?  ?  20
?  ?  7  3  ?  ?  22 ?  ?  13 ?  ?  21 ?  ?  5  ?  ?  17 18 25 12 4  6  2  30 11 35 20 ?  ?  8  16 ?  24 34
25 ?  ?  ?  4  ?  ?  8  ?  ?  33 20 ?  3  31 ?  ?  ?  ?  28 29 ?  5  ?  16 ?  7  ?  18 21 22 13 ?  ?  ?  26
?  ?  2  ?  ?  ?  ?  26 16 31 ?  15 7  6  33 1  8  ?  ?  ?  14 ?  ?  9  28 22 ?  5  ?  29 12 ?  11 ?  30 ?
29 23 ?  15 11 ?  ?  ?  27 25 ?  ?  22 16 ?  20 4  35 26 ?  ?  36 ?  7  ?  ?  13 ?  34 24 21 2  ?  ?  33 6
?  ?  ?  22 ?  32 ?  10 4  29 5  ?  25 ?  ?  27 9  ?  ?  ?  23 35 30 ?  ?  17 26 ?  33 ?  36 15 28 ?  ?  7
16 15 ?  2  ?  ?  35 ?  6  ?  26 13 9  24 20 ?  27 11 ?  ?  ?  1  31 ?  ?  ?  21 3  ?  ?  ?  ?  ?  5  ?  17
?  1  34 30 ?  ?  ?  33 ?  9  ?  16 ?  ?  ?  ?  21 6  ?  15 ?  22 ?  ?  ?  12 10 14 5  31 2  ?  27 26 18 ?
?  18 ?  5  24 ?  20 1  ?  17 23 ?  ?  34 8  33 ?  ?  30 ?  ?  ?  13 ?  ?  25 ?  22 ?  ?  10 ?  31 28 11 ?
6  21 25 32 ?  20 ?  28 ?  ?  ?  3  ?  17 ?  31 ?  ?  23 5  ?  ?  ?  ?  ?  ?  1  ?  7  8  ?  24 ?  ?  16 ?
?  9  13 29 10 28 11 22 ?  ?  25 4  16 ?  14 12 1  7  3  36 ?  ?  ?  ?  30 ?  ?  15 27 ?  ?  34 ?  6  ?  33
?  31 22 ?  8  3  15 7  ?  ?  18 ?  ?  ?  ?  ?  ?  19 34 6  17 11 ?  12 ?  ?  ?  23 29 36 35 ?  ?  32 ?  ?

// http://www.menneske.no/sudoku/6/eng/showpuzzle.html?number=230
```

Does GAC Matter?

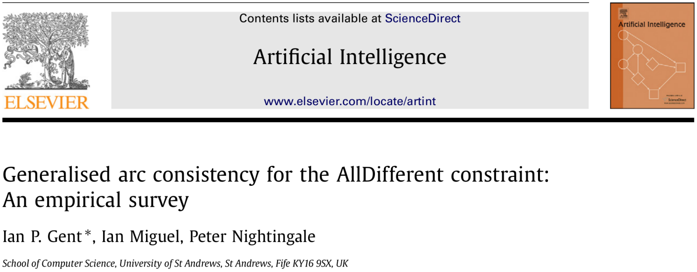
```
Limit reached.
Model[Sudoku]
Solutions: 0
Building time : 0.110s
Resolution time : 3,600.002s
Nodes: 9,037,226 (2,510.3 n/s)
Backtracks: 18,074,338
Fails: 9,037,183
Restarts: 0
```

Does GAC Matter?

```
Limit reached.
Model[Sudoku]
Solutions: 0
Building time : 0.110s
Resolution time : 3,600.002s
Nodes: 9,037,226 (2,510.3 n/s)
Backtracks: 18,074,338
Fails: 9,037,183
Restarts: 0
```

```
1 solution found.
Model[Sudoku]
Solutions: 1
Building time : 0.062s
Resolution time : 0.203s
Nodes: 28 (137.6 n/s)
Backtracks: 48
Fails: 26
Restarts: 0
```

Implementing AllDifferent


Contents lists available at ScienceDirect

Artificial Intelligence

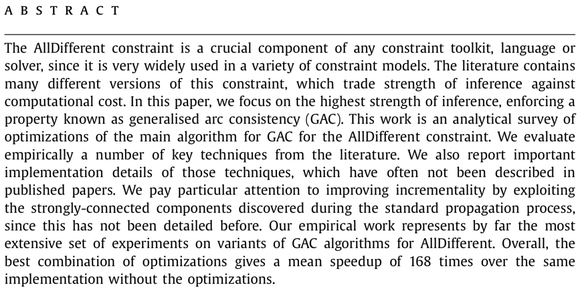
www.elsevier.com/locate/artint

Generalised arc consistency for the AllDifferent constraint: An empirical survey

Ian P. Gent*, Ian Miguel, Peter Nightingale

School of Computer Science, University of St Andrews, St Andrews, Fife KY16 9SX, UK

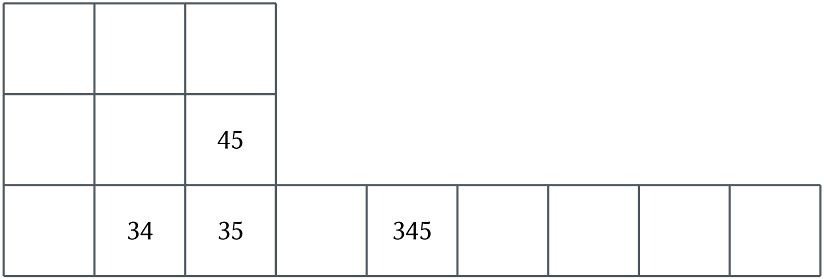
Implementing AllDifferent



A B S T R A C T

The AllDifferent constraint is a crucial component of any constraint toolkit, language or solver, since it is very widely used in a variety of constraint models. The literature contains many different versions of this constraint, which trade strength of inference against computational cost. In this paper, we focus on the highest strength of inference, enforcing a property known as generalised arc consistency (GAC). This work is an analytical survey of optimizations of the main algorithm for GAC for the AllDifferent constraint. We evaluate empirically a number of key techniques from the literature. We also report important implementation details of those techniques, which have often not been described in published papers. We pay particular attention to improving incrementality by exploiting the strongly-connected components discovered during the standard propagation process, since this has not been detailed before. Our empirical work represents by far the most extensive set of experiments on variants of GAC algorithms for AllDifferent. Overall, the best combination of optimizations gives a mean speedup of 168 times over the same implementation without the optimizations.

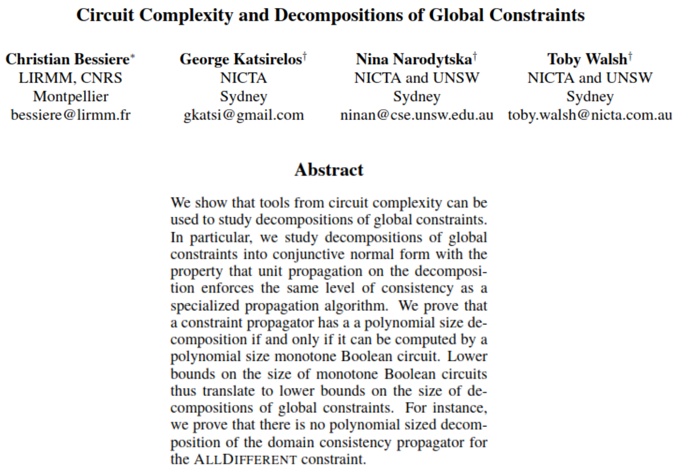
Are You Smarter than a Constraint Solver?



Are You Smarter than a Constraint Solver?

- Propagation only considers *one constraint at a time*, and the only communication between constraints is by deleting values.
- There are various ways of automatically combining two constraints.
  - But getting "the best possible" filtering from two "all different" constraints simultaneously is NP-hard. . .

You Can’t Do This in CNF



Circuit Complexity and Decompositions of Global Constraints

Christian Bessiere[*]
LIRMM, CNRS
Montpellier
bessiere@lirmm.fr

George Katsirelos[†]
NICTA
Sydney
gkatsi@gmail.com

Nina Narodytska[†]
NICTA and UNSW
Sydney
ninan@cse.unsw.edu.au

Toby Walsh[†]
NICTA and UNSW
Sydney
toby.walsh@nicta.com.au

Abstract

We show that tools from circuit complexity can be used to study decompositions of global constraints. In particular, we study decompositions of global constraints into conjunctive normal form with the property that unit propagation on the decomposition enforces the same level of consistency as a specialized propagation algorithm. We prove that a constraint propagator has a a polynomial size decomposition if and only if it can be computed by a polynomial size monotone Boolean circuit. Lower bounds on the size of monotone Boolean circuits thus translate to lower bounds on the size of decompositions of global constraints. For instance, we prove that there is no polynomial sized decomposition of the domain consistency propagator for the ALLDIFFERENT constraint.

Proof Logging for CP

- CP needs this: solvers are a little bit buggy.
- Hard to do, though:
  - Practical SAT encodings are buggy, and can't tackle many problems.
  - Propagators do strong reasoning, can't justify all-different practically in DRAT.
  - Need a proof format that's simple to verify.

Proof Logging for CP

- CP needs this: solvers are a little bit buggy.
- Hard to do, though:
  - Practical SAT encodings are buggy, and can't tackle many problems.
  - Propagators do strong reasoning, can't justify all-different practically in DRAT.
  - Need a proof format that's simple to verify.
- Amazing recent progress: pseudo-Boolean proof logging is enough!

Constraint-Based Local Search

- Use constraints and high level structure to define neighbourhoods.
- Good at coping with hard constraints.
- Does not satisfy Karam's definition of local search.

Why Isn't My Problem Satisfiable?

- Need human-understandable explanations of unsatisfiability.
- Use high-level constraints to describe cores?
- Minimal unsatisfiable subsets?

Belief Propagation

$$x_i \in \{1, 2, 3, 4\} \qquad \mathit{alldifferent}(x_1, x_2, x_3)$$
$$x_1 + x_2 + x_3 + x_4 = 7 \qquad x_3 \leq x_4$$

- Use weighted model counting on constraints to get rough solution frequencies for each value.
- Update weights, iterate a few times.
- Needs high-level constraints, not decompositions.

	1	2	3	4
x_1	.01	.52	.46	.01
x_2	.01	.52	.46	.01
x_3	.98	.02	.00	.00
x_4	.90	.10	.00	.00

Decision Diagram Solvers

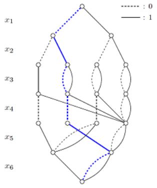


Figure 1: Exact BDD

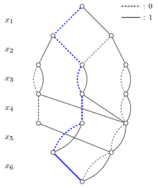


Figure 2: Relaxed BDD

- Turn the search tree into a DAG.
- Merge identical and dominating states.
- Merge non-identical states to get lower and upper bounds.

Parallel Search

- Work stealing: scalability limits, erratic behaviour.
- Confidence-based work stealing: scalability limits, hard to implement.
- Embarrassingly parallel search: easy to implement, works very well for some problems.

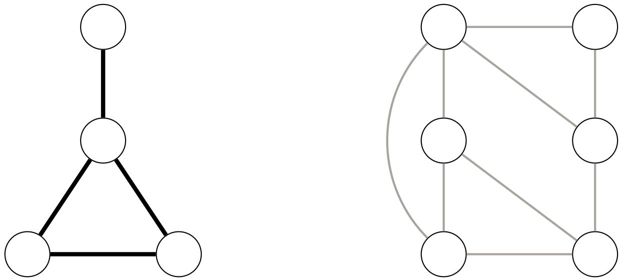
Parallel Search

- Work stealing: scalability limits, erratic behaviour.
- Confidence-based work stealing: scalability limits, hard to implement.
- Embarrassingly parallel search: easy to implement, works very well for some problems.
- Abusing restarts and value-ordering heuristics: easy to implement, scales very well, respects search order.

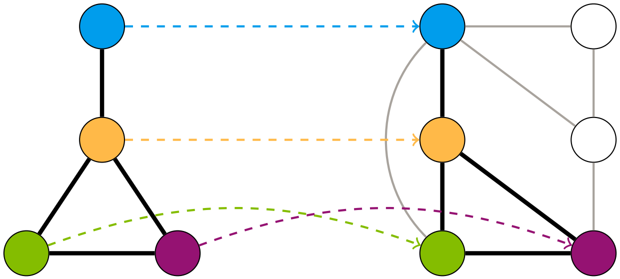
Competitions

- Don’t cover many solver features.
- Not the primary driving force behind evaluating solvers.
- Many solvers don’t compete or aren’t eligible.
- This is both a Bad Thing and a Good Thing.

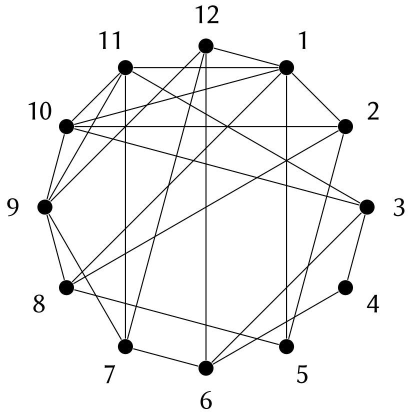
Subgraph Isomorphism



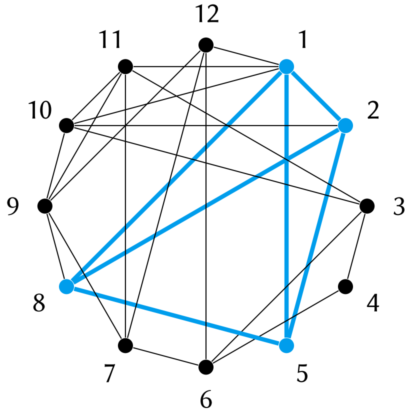
Subgraph Isomorphism



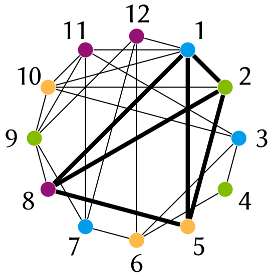
The Maximum Clique Problem



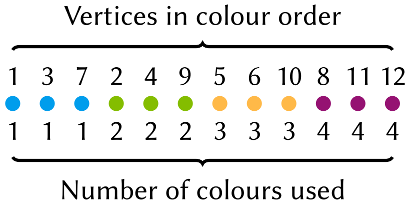
The Maximum Clique Problem



Colour Ordering



$\rightsquigarrow$



Colour Ordering

- Vertices in the rightmost colour class are “generally expected [to have a] high probability of belonging to a maximum clique” according to Tomita and Kameda, J. Global Optimization, 37(1) 2007.

Colour Ordering

- Vertices in the rightmost colour class are “generally expected [to have a] high probability of belonging to a maximum clique” according to Tomita and Kameda, J. Global Optimization, 37(1) 2007.
- It’s not true.

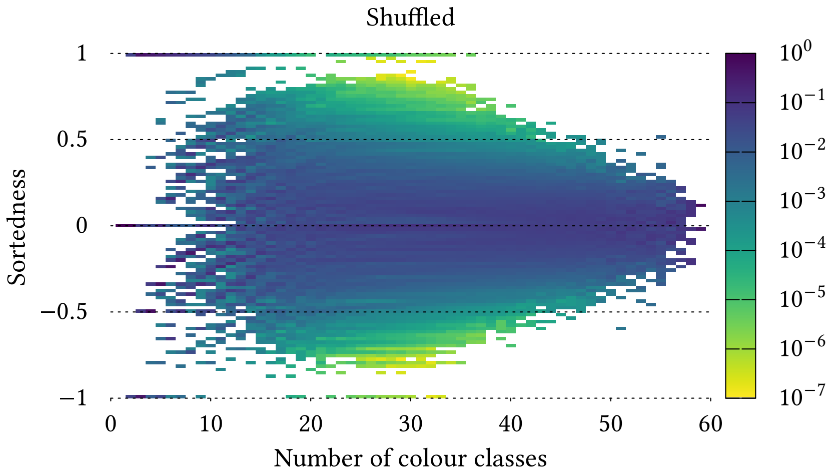
Colour Ordering

- Vertices in the rightmost colour class are "generally expected [to have a] high probability of belonging to a maximum clique" according to Tomita and Kameda, J. Global Optimization, 37(1) 2007.
- It's not true.
- Right to left is still better even if the algorithm is only proving optimality.
- Better clique algorithms have *worse* anytime behaviour and take *longer* to find a strong incumbent.

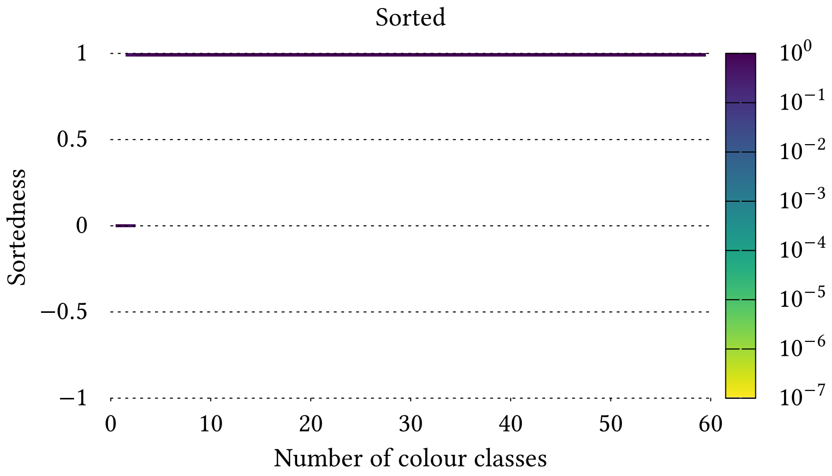
A Hypothesis: Smallest Domain First?

- Branching on a colour class is like branching on a domain in CP, where the values are the vertices in the colour class plus a null value.
- Smallest domain first is a good heuristic.
- Greedy colourings tend to produce larger colour classes first.
- Right to left is smallest domain first.

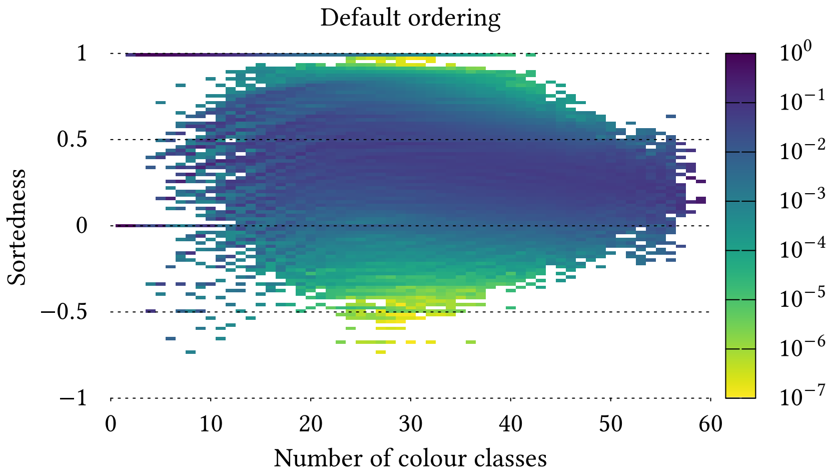
We Can Measure This!



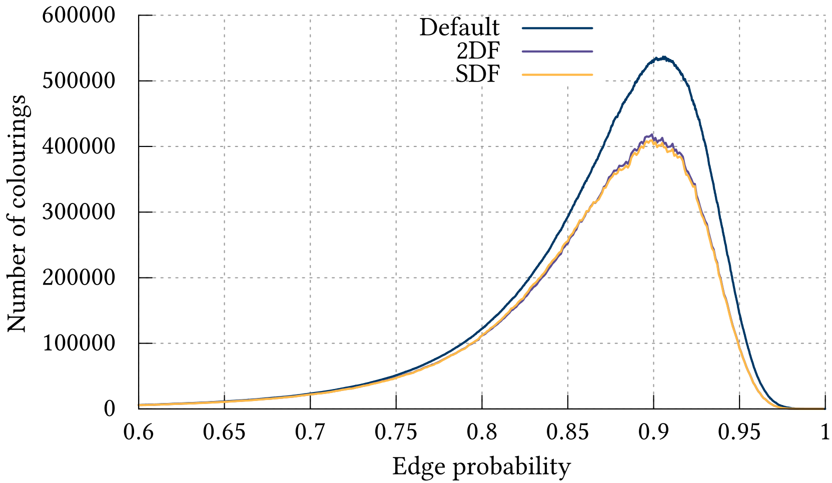
We Can Measure This!



We Can Measure This!



Increasing Sortedness Decreases Search Space Size



However…

- Small impact on runtime.
- Hard to sell “understanding why this algorithm works” compared to “this new algorithm is better”.

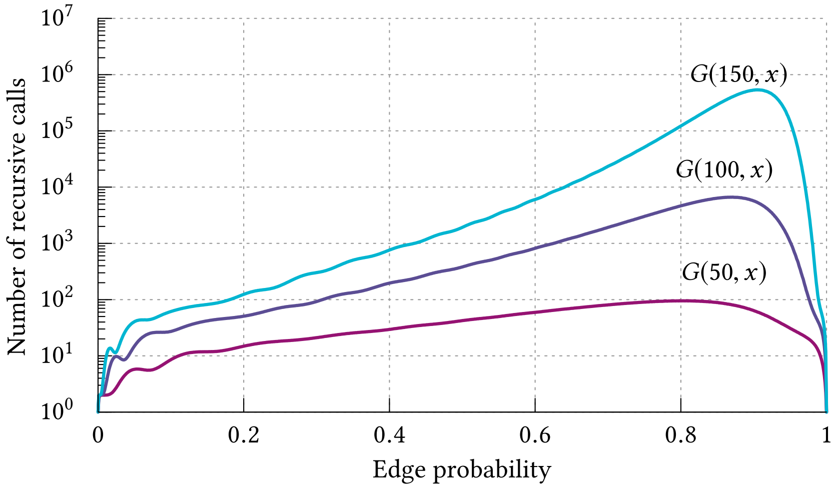
Cliques in Random Graphs

- Science on SAT solvers on industrial instances is like asking for biochemistry when all we have is alchemy.
- Let’s aim for simple 1800s chemistry experiments.
- Simple solver, simple instances.

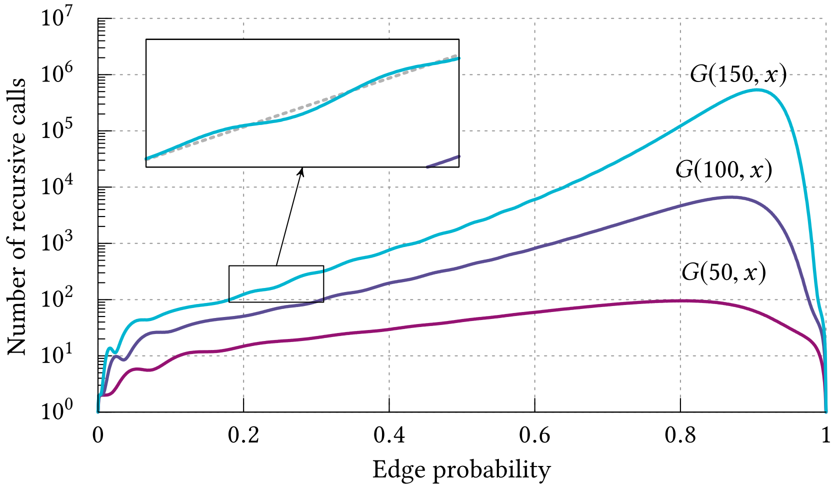
Cliques in Random Graphs

- Science on SAT solvers on industrial instances is like asking for biochemistry when all we have is alchemy.
- Let’s aim for simple 1800s chemistry experiments.
- Simple solver, simple instances.
- Very accurate measurements, huge sample size.

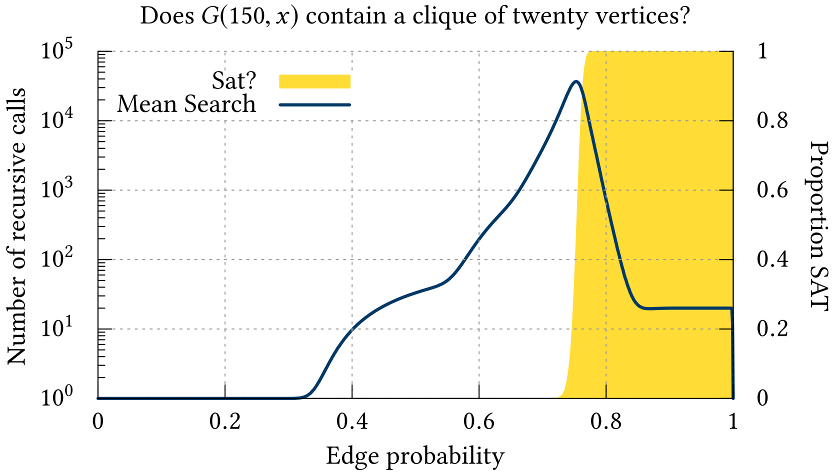
Cliques in Random Graphs



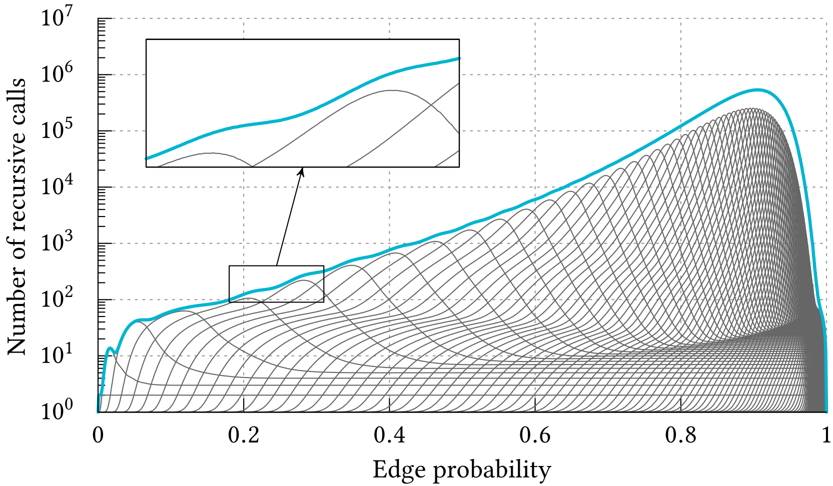
Cliques in Random Graphs



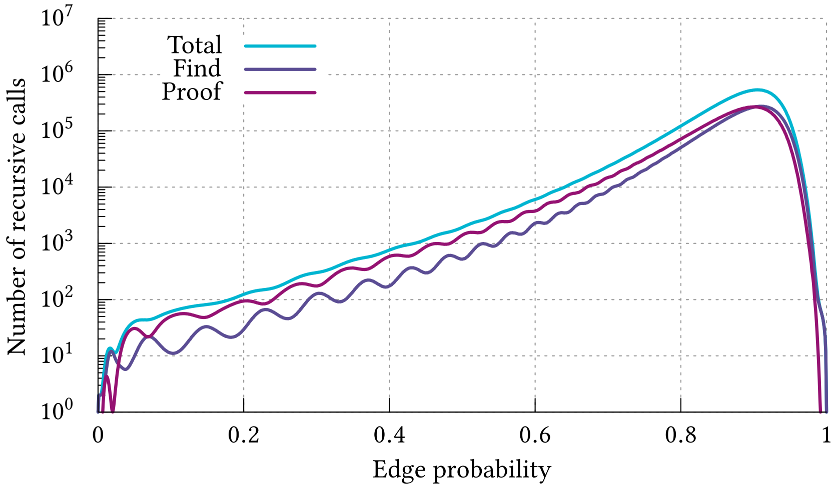
Cliques in Random Graphs



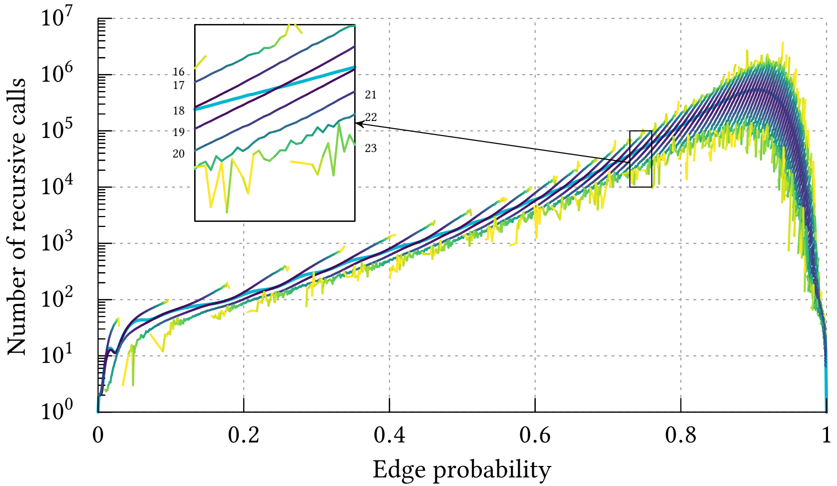
Cliques in Random Graphs



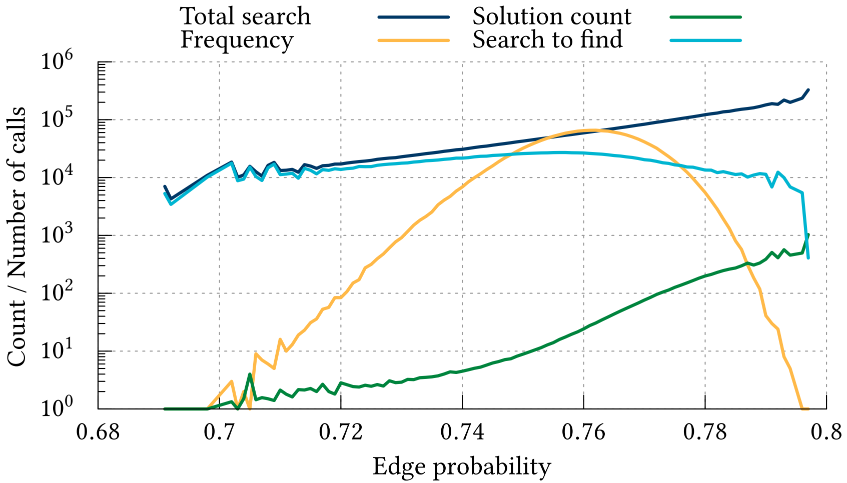
Cliques in Random Graphs



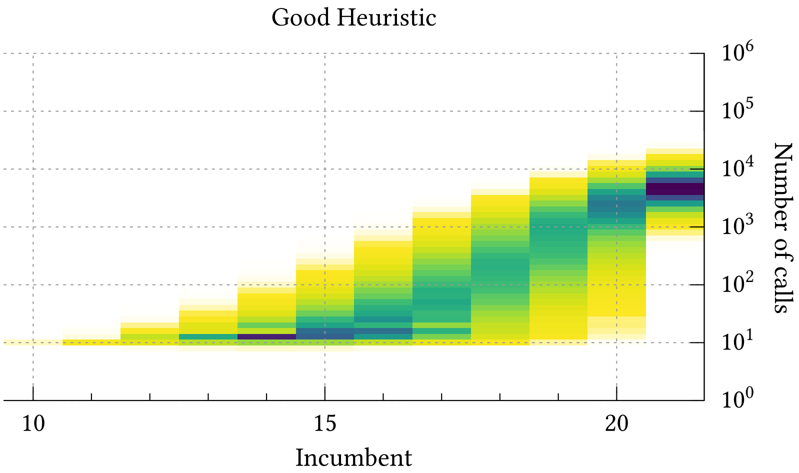
Cliques in Random Graphs



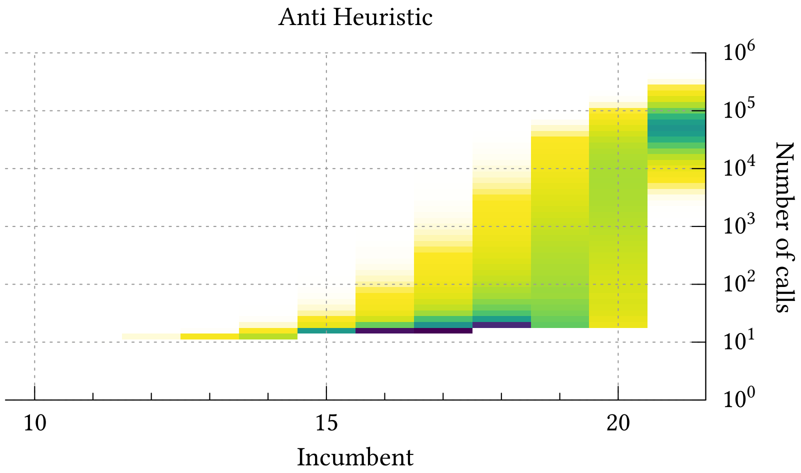
Cliques in Random Graphs



Cliques in Random Graphs



Cliques in Random Graphs



https://ciaranm.github.io/

ciaran.mccreesh@glasgow.ac.uk