The "All Different" Constraint

Ciaran McCreesh and Patrick Prosser

Global Constraints

- A *global constraint* is one which can operate on arbitrarily many variables.

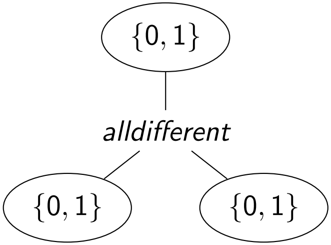
Two-Colouring a Triangle

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

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

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

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



Decomposing “All Different”

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

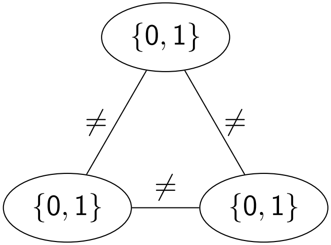
$x_2 \in \{0, 1\}$

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

$x_1 \neq x_2$

$x_1 \neq x_3$

$x_2 \neq x_3$



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.

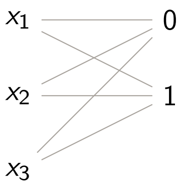
What Would a Human Do?

“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.

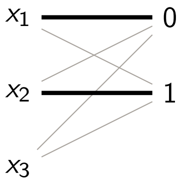
Matchings

- 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 (see Algorithmics II).
- There is a matching which covers each variable if and only if the constraint can be satisfied.



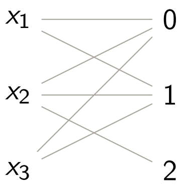
Matchings

- 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 (see Algorithmics II).
- There is a matching which covers each variable if and only if the constraint can be satisfied.



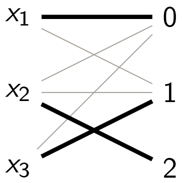
Matchings

- 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 (see Algorithmics II).
- There is a matching which covers each variable if and only if the constraint can be satisfied.



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 (see Algorithmics II).
- There is a matching which covers each variable if and only if the constraint can be satisfied.

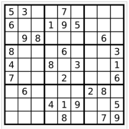


Sudoku

From Wikipedia, the free encyclopedia

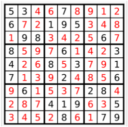
Not to be confused with Sodoku or Sudeki.

Sudoku (数独 *sūdoku*?, digit-single) /suːˈdoʊkuː/, /-ˈdɒ-/, /sə-/; originally called **Number Place**,[1] is a logic-based,[2][3] combinatorial[4] number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid (also called "boxes", "blocks", "regions", or "sub-squares") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a unique solution.



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

A typical Sudoku puzzle



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

The same puzzle with solution numbers marked in red

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

What Does Choco Do?

```
Model model = new Model("one␣row␣sudoku");
Solver solver = model.getSolver();
IntVar[] row = new IntVar[9];
row[0] = model.intVar(new int[]{1,8});
row[1] = model.intVar(new int[]{2,3});
row[2] = model.intVar(new int[]{2,3});
row[3] = model.intVar(new int[]{2,4,5});
row[4] = model.intVar(new int[]{4,5,6});
row[5] = model.intVar(new int[]{4,5,6});
row[6] = model.intVar(new int[]{2,7,9});
row[7] = model.intVar(new int[]{3,7,8});
row[8] = model.intVar(new int[]{2,3,5,8,9});
System.out.println("Before:␣" + Arrays.toString(row));

model.allDifferent(row).post();
solver.propagate();
System.out.println("After:␣" + Arrays.toString(row));
```

What Does Choco Do?

```
Before:
  [IV_1 = {1,8},     IV_2 = {2..3},    IV_3 = {2..3},
   IV_4 = {2,4..5}, IV_5 = {4..6},    IV_6 = {4..6},
   IV_7 = {2,7,9},   IV_8 = {3,7..8}, IV_9 = {2..3,5,8..9}]

After:
  [IV_1 = 1,         IV_2 = {2..3},    IV_3 = {2..3},
   IV_4 = {4..5},    IV_5 = {4..6},    IV_6 = {4..6},
   IV_7 = {7,9},     IV_8 = {7..8},    IV_9 = {8..9}]
```

1	23	23	45	456	456	79	78	89

What Does Choco Do?

```
Model model = new Model("one␣row␣sudoku");
Solver solver = model.getSolver();
IntVar[] row =  new IntVar[9];
row[0] = model.intVar(new int[]{1,8});
row[1] = model.intVar(new int[]{2,3});
row[2] = model.intVar(new int[]{2,3});
row[3] = model.intVar(new int[]{2,4,5});
row[4] = model.intVar(new int[]{4,5,6});
row[5] = model.intVar(new int[]{4,5,6});
row[6] = model.intVar(new int[]{2,7,9});
row[7] = model.intVar(new int[]{3,7,8});
row[8] = model.intVar(new int[]{2,3,5,8,9});
System.out.println("Before:␣" + Arrays.toString(row));

for (int i = 0 ; i < 8 ; ++i)
    for (int j = i + 1 ; j < 9 ; ++j)
        model.arithm(row[i], "!=", row[j]).post();

solver.propagate();
System.out.println("After:␣" + Arrays.toString(row));
```

What Does Choco Do?

```
Before:
  [IV_1 = {1,8},     IV_2 = {2..3},    IV_3 = {2..3},
   IV_4 = {2,4..5},  IV_5 = {4..6},    IV_6 = {4..6},
   IV_7 = {2,7,9},   IV_8 = {3,7..8},  IV_9 = {2..3,5,8..9}]

After (neq):
  [IV_1 = {1,8},     IV_2 = {2..3},    IV_3 = {2..3},
   IV_4 = {2,4..5},  IV_5 = {4..6},    IV_6 = {4..6},
   IV_7 = {2,7,9},   IV_8 = {3,7..8},  IV_9 = {2..3,5,8..9}]

After (all different):
  [IV_1 = 1,         IV_2 = {2..3},    IV_3 = {2..3},
   IV_4 = {4..5},    IV_5 = {4..6},    IV_6 = {4..6},
   IV_7 = {7,9},     IV_8 = {7..8},    IV_9 = {8..9}]
```

Generalised Arc Consistency

- Arc Consistency: for a binary constraint, each value is supported by at least one value in the other variable.
- Generalised Arc Consistency: for a global constraint, we can pick any value from any variable, and find a supporting set of values from each other variable in the constraint simultaneously.

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.
- If we do this for 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.

Only Occurs in One Place?

"But wait! We said that the value 1 only occurs in one place. That doesn't sound like a Hall set!"

Only Occurs in One Place?

“But wait! We said that the value 1 only occurs in one place. That doesn’t sound like a Hall set!”

- The “only occurs in one place” rule we used first is just a Hall set of size 8.

Is Choco Magic?

- There are 2^n potential Hall sets, so considering them all is probably a bad idea. However, there is a polynomial algorithm.
- This algorithm isn't examinable, but here's an animation of roughly how it works, so you don't have to believe it's magic any more.

Is Choco Magic?

18	23	23	245	456	456	279	378	23589

Is Choco Magic?

18	23	23	245	456	456	279	378	23589

$row[0]$

$row[1]$

$row[2]$

$row[3]$

$row[4]$

$row[5]$

$row[6]$

$row[7]$

$row[8]$

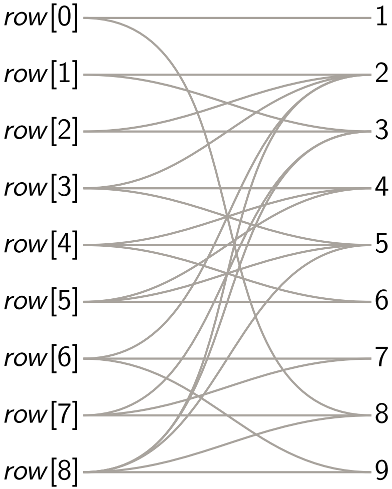
Is Choco Magic?

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

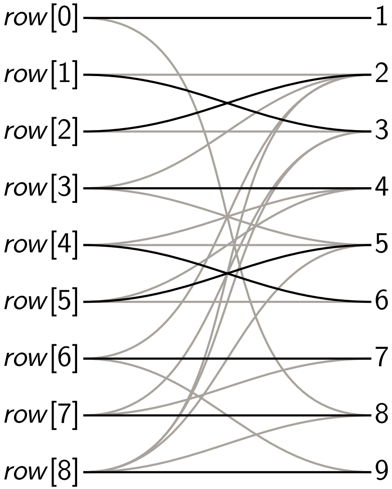
Is Choco Magic?

18	23	23	245	456	456	279	378	23589



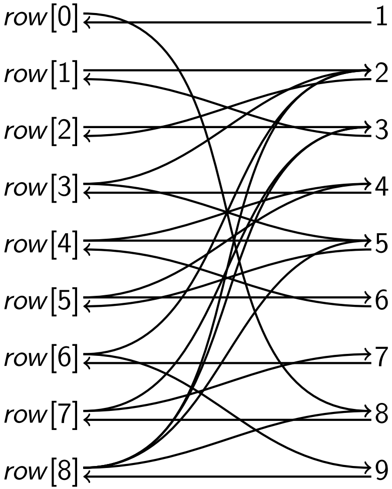
Is Choco Magic?

18	23	23	245	456	456	279	378	23589



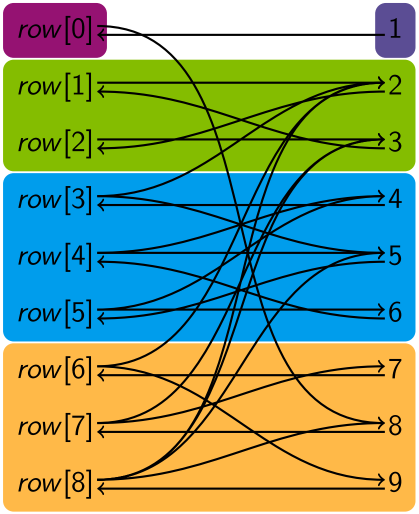
Is Choco Magic?

18	23	23	245	456	456	279	378	23589



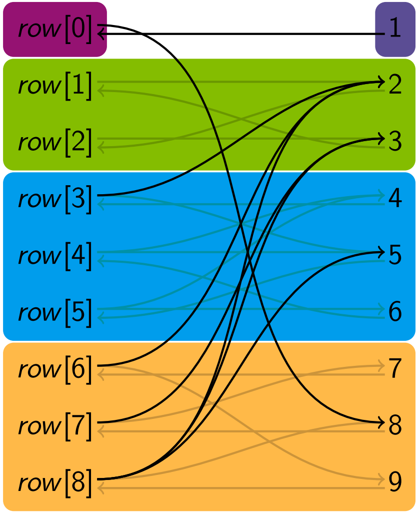
Is Choco Magic?

18	23	23	245	456	456	279	378	23589



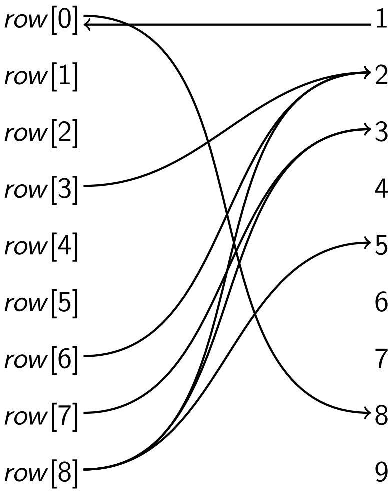
Is Choco Magic?

18	23	23	245	456	456	279	378	23589



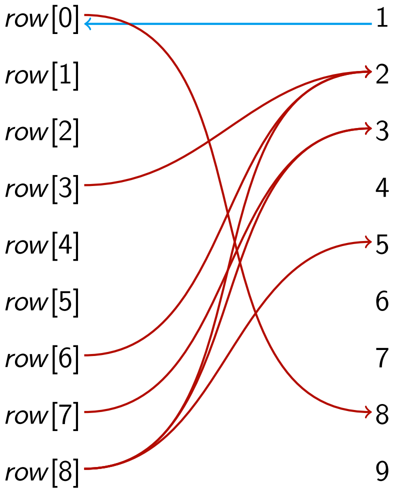
Is Choco Magic?

18	23	23	245	456	456	279	378	23589



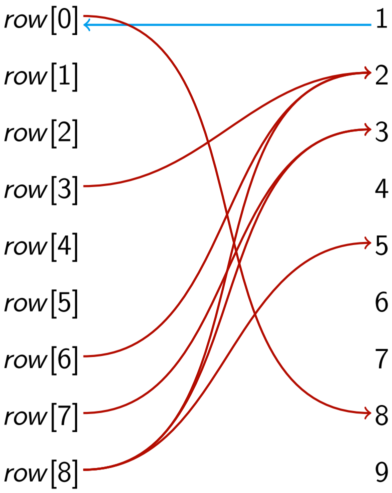
Is Choco Magic?

18	23	23	245	456	456	279	378	23589



Is Choco Magic?

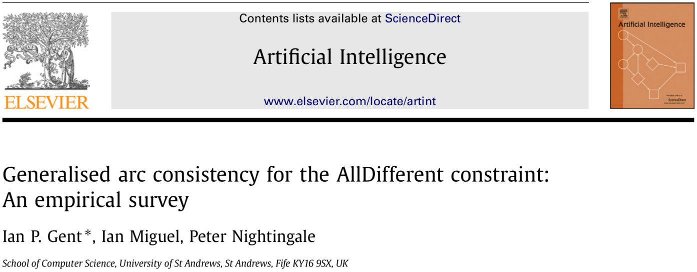
18	23	23	245	456	456	279	378	23589



Is Choco Magic?

- There's one more special condition that can happen, if we have more values than domains.

Implementing AllDifferent



Contents lists available at ScienceDirect

Artificial Intelligence

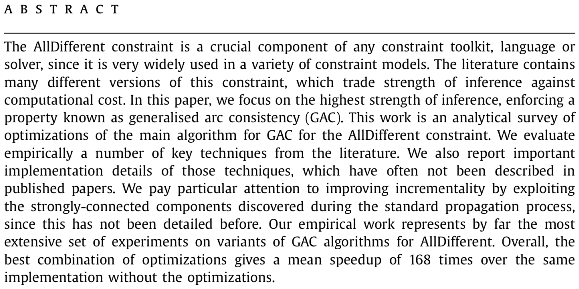
www.elsevier.com/locate/artint

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.

A Sudoku Solver in Choco

```
0 0 3 1 2 0 0 9 0
1 0 2 0 0 3 6 0 0
7 0 0 9 6 8 2 1 0
0 0 0 8 0 0 7 0 0
6 0 5 4 7 1 8 0 0
0 8 0 0 0 9 5 0 0
0 0 6 7 1 2 0 0 0
0 0 0 0 0 0 0 0 6
2 1 8 0 9 5 0 7 4

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

A Sudoku Solver in Choco

```
int n = 3;
int nn = n * n;
int[][] predef = new int[nn][nn];

try (Scanner sc = new Scanner(new File(args[0]))) {
    for (int i = 0 ; i < nn ; i++)
        for (int j = 0 ; j < nn ; j++)
            predef[i][j] = sc.nextInt();
}
```

A Sudoku Solver in Choco

```
Model model = new Model("Sudoku");
IntVar[][] grid = model.intVarMatrix("grid", nn, nn, 1, nn);
```

A Sudoku Solver in Choco

```
// Rows
for (int i = 0 ; i < nn ; ++i)
    model.allDifferent(grid[i]).post();
```

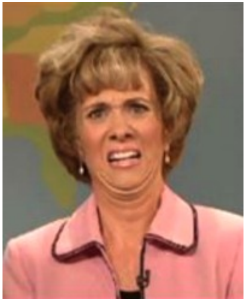
A Sudoku Solver in Choco

```
// Columns
for (int i = 0 ; i < nn ; ++i) {
    IntVar[] column = new IntVar[nn];
    for (int j = 0 ; j < nn ; ++j)
        column[j] = grid[j][i];
    model.allDifferent(column).post();
}
```

A Sudoku Solver in Choco

```
// Squares
for (int i = 0 ; i < nn ; i += n)
    for (int j = 0 ; j < nn ; j += n) {
        IntVar[] square = new IntVar[nn];
        for (int x = 0 ; x < n ; ++x)
            for (int y = 0 ; y < n ; ++y)
                square[n * x + y] = grid[i + x][j + y];
        model.allDifferent(square).post();
    }
```

A Sudoku Solver in Choco



```
// Squares
for (int i = 0 ; i < nn ; i += n)
    for (int j = 0 ; j < nn ; j += n) {
        IntVar[] square = new IntVar[nn];
        for (int x = 0 ; x < n ; ++x)
            for (int y = 0 ; y < n ; ++y)
                square[n * x + y] = grid[i + x][j + y];
        model.allDifferent(square).post();
    }
```

A Sudoku Solver in Choco

```
// Predefined values
for (int i = 0 ; i < nn ; i++)
    for (int j = 0 ; j < nn ; j++)
        if (0 != predef[i][j])
            model.arithm(grid[i][j], "=", predef[i][j]).post();
```

A Sudoku Solver in Choco

```
if (model.getSolver().solve()) {
    for (int i = 0 ; i < nn ; i++) {
        for (int j = 0 ; j < nn ; j++)
            System.out.print(grid[i][j].getValue() + "␣");
        System.out.println();
    }
}

System.out.println("\n" + model.getSolver().getMeasures());
```

Some Experiments

```
0 0 3 1 2 0 0 9 0
1 0 2 0 0 3 6 0 0
7 0 0 9 6 8 2 1 0
0 0 0 8 0 0 7 0 0
6 0 5 4 7 1 8 0 0
0 8 0 0 0 9 5 0 0
0 0 6 7 1 2 0 0 0
0 0 0 0 0 0 0 0 6
2 1 8 0 9 5 0 7 4

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

Some Experiments

```
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
```

Some Experiments

```
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
```

Some Experiments

```
0 0 6 3 0 0 0 0 1
9 0 0 0 0 0 6 0 0
0 7 0 0 0 0 0 5 0
0 0 0 2 0 1 0 0 0
3 5 0 0 9 0 0 2 0
0 0 0 5 0 0 0 0 0
0 4 8 0 0 0 0 1 0
0 6 0 0 0 0 0 0 0
0 0 1 0 0 6 3 7 8

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

Some Experiments

```
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
```

Some Experiments

```
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
```

Some Experiments

```
9 0 0 0 5 0 0 0 4
0 7 0 0 0 6 1 0 0
0 0 0 0 0 0 8 3 0
0 0 0 0 8 1 0 2 0
2 0 0 5 0 3 0 0 8
0 9 0 2 7 0 0 0 0
0 3 6 0 0 0 0 0 0
0 0 2 3 0 0 0 7 0
5 0 0 0 2 0 0 0 6

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

Some Experiments

```
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
```

Some Experiments

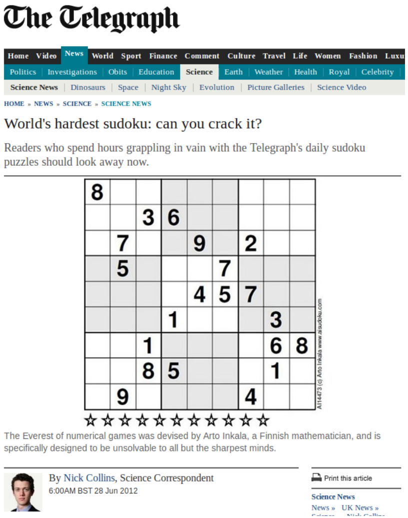
```
9 8 3 1 5 2 7 6 4
4 7 5 8 3 6 1 9 2
6 2 1 9 4 7 8 3 5
3 5 4 6 8 1 9 2 7
2 6 7 5 9 3 4 1 8
1 9 8 2 7 4 6 5 3
7 3 6 4 1 5 2 8 9
8 4 2 3 6 9 5 7 1
5 1 9 7 2 8 3 4 6

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
```

Some Experiments



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

Print this article

Science News
News » UK News »

Some Experiments

```
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
```

Some Experiments

```
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
```

Some Experiments

```
0  0  15 0  0  34 28 18 21 0  0  0  27 0  36 29 5  14 4  0  0  0  0  0  0  23 0  0  24 10 26 32 0  12 19 0
23 0  6  0  2  0  0  20 32 0  0  33 0  0  0  0  26 13 22 9  5  21 0  8  17 29 0  0  36 1  15 28 7  35 31 0
0  26 0  0  14 0  27 36 0  7  0  0  0  0  0  0  6  28 0  0  34 0  23 0  3  0  0  0  30 0  16 0  18 9  8  29
0  32 9  28 0  16 0  0  11 0  24 0  0  33 0  0  0  21 0  0  19 10 14 0  0  8  2  0  12 7  0  27 17 0  25 0
0  4  31 17 0  8  23 35 10 16 6  0  0  0  1  0  30 0  29 26 0  0  0  0  9  0  33 0  14 0  0  0  13 36 34 0
5  0  33 0  0  0  0  0  13 2  0  0  0  8  32 0  0  0  15 3  0  7  12 36 34 27 0  19 0  26 0  0  23 0  10 14
20 0  0  8  23 26 0  21 0  1  19 0  0  32 16 2  0  0  0  12 22 0  0  28 0  18 30 10 11 0  6  0  15 0  0  0
35 11 0  0  13 25 36 4  0  15 0  0  28 0  6  0  0  1  31 23 26 0  3  18 0  0  9  29 0  0  0  17 33 0  2  30
0  7  0  4  0  9  5  0  17 0  11 24 31 0  0  30 0  0  0  35 8  29 19 32 23 0  34 20 6  0  0  0  0  1  0  0
18 0  0  0  22 33 29 27 0  26 0  2  0  9  0  11 14 0  0  0  0  17 16 0  35 19 0  0  1  0  0  12 0  0  0  31
3  16 0  36 32 0  0  6  30 10 8  34 5  18 27 22 17 23 0  0  7  0  0  1  0  0  4  0  0  28 0  0  29 24 0  0
21 0  0  31 0  0  33 0  0  14 0  0  0  0  29 26 0  4  0  2  36 0  0  0  32 0  0  27 0  5  13 19 0  18 0  0
34 17 0  0  9  0  2  0  0  20 13 10 0  0  0  18 7  0  0  0  0  0  0  22 6  0  32 0  15 0  19 0  0  0  4  0
24 0  0  7  25 15 1  23 3  0  31 0  26 0  19 14 0  2  21 34 6  0  0  0  20 0  0  8  17 30 33 22 36 0  0  10
11 27 0  33 6  1  0  34 25 22 0  8  36 0  9  0  23 24 0  19 31 3  0  26 21 0  0  0  4  0  29 18 30 15 0  0
0  0  0  14 0  0  0  32 0  0  4  26 0  15 0  0  0  20 9  0  0  0  29 30 0  0  16 0  22 0  0  7  0  0  0  0
32 0  18 0  0  0  0  5  0  0  36 0  0  0  35 21 0  12 0  16 0  20 0  27 0  0  3  0  9  0  1  26 8  2  0  0
36 0  0  0  16 0  7  0  18 0  0  21 0  0  22 0  0  17 35 14 0  24 25 0  29 2  0  0  0  0  31 0  3  0  6  9
30 29 0  12 0  5  0  0  0  0  20 17 0  11 21 0  35 9  28 0  0  18 0  0  1  0  0  34 0  23 0  36 0  0  0  27
0  0  11 13 3  27 0  16 0  24 0  0  17 0  7  0  33 0  5  0  35 6  0  0  0  10 0  0  25 0  0  0  0  30 0  21
0  0  0  0  28 0  0  13 0  5  0  0  10 1  0  0  0  31 11 0  0  0  22 0  19 21 0  0  2  0  0  0  9  0  0  0
0  0  17 25 20 23 0  30 0  0  0  1  13 0  24 16 29 0  7  21 0  27 0  10 14 0  35 9  28 0  11 4  2  0  26 12
9  0  0  34 18 2  25 12 14 0  0  35 0  0  0  6  19 27 8  0  1  16 0  3  0  4  0  7  31 11 17 20 24 0  0  22
0  24 0  0  0  35 0  15 0  6  0  11 12 0  0  0  0  0  0  25 20 0  0  0  33 36 29 0  0  27 0  23 0  0  7  28
0  0  14 0  5  18 24 0  34 0  0  7  0  0  0  23 12 0  10 0  11 8  0  0  0  0  6  0  0  4  0  0  32 0  0  20
0  0  7  3  0  0  22 0  0  13 0  0  21 0  0  5  0  0  17 18 25 12 4  6  2  30 11 35 20 0  0  8  16 0  24 34
25 0  0  0  4  0  0  8  0  0  33 20 0  3  31 0  0  0  0  28 29 0  5  0  16 0  7  0  18 21 22 13 0  0  0  26
0  0  2  0  0  0  0  26 16 31 0  15 7  6  33 1  8  0  0  0  14 0  0  9  28 22 0  5  0  29 12 0  11 0  30 0
29 23 0  15 11 0  0  0  27 25 0  0  22 16 0  20 4  35 26 0  0  36 0  7  0  0  13 0  34 24 21 2  0  0  33 6
0  0  0  22 0  32 0  10 4  29 5  0  25 0  0  27 9  0  0  0  23 35 30 0  0  17 26 0  33 0  36 15 28 0  0  7
16 15 0  2  0  0  35 0  6  0  26 13 9  24 20 0  27 11 0  0  0  1  31 0  0  0  21 3  0  0  0  0  0  5  0  17
0  1  34 30 0  0  0  33 0  9  0  16 0  0  0  0  21 6  0  15 0  22 0  0  0  12 10 14 5  31 2  0  27 26 18 0
0  18 0  5  24 0  20 1  0  17 23 0  0  34 8  33 0  0  30 0  0  0  13 0  0  25 0  22 0  0  10 0  31 28 11 0
6  21 25 32 0  20 0  28 0  0  0  3  0  17 0  31 0  0  23 5  0  0  0  0  0  0  1  0  7  8  0  24 0  0  16 0
0  9  13 29 10 28 11 22 0  0  25 4  16 0  14 12 1  7  3  36 0  0  0  0  30 0  0  15 27 0  0  34 0  6  0  33
0  31 22 0  8  3  15 7  0  0  18 0  0  0  0  0  0  19 34 6  17 11 0  12 0  0  0  23 29 36 35 0  0  32 0  0

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

Some Experiments

```
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
```

Some Experiments

```
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
```

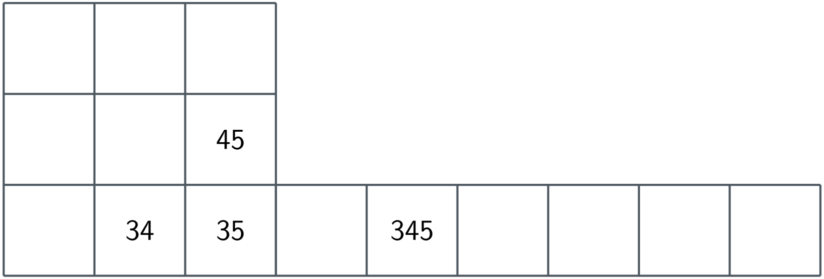
Do Global Constraints Always Help?

- Sometimes globals make a spectacular difference.
- Sometimes global constraints end up not giving any more deletions than their decompositions, and can take longer to propagate. Sometimes extra deletions don't help anyway.
- Some global constraints cannot be *decomposed*. Any global constraint can be *encoded* using binary constraints in a very unpleasant way involving polynomially many extra variables, but AC on an encoding can be weaker than GAC.
- Some global constraints only have weaker propagators: GAC can be too hard to be practical, or NP-complete on its own.
- Using globals isn't a *guaranteed* benefit, but they make the model easier to read, and it's easier to translate from globals to decompositions and encodings than the other way around.

Generalisations of All Different

- All different except 0.
- Global cardinality.
- At least, at most, among.
- *n* Value.

Are You Smarter than a Constraint Solver?



Are You Smarter than a Constraint Solver?

- Remember: propagation only considers *one constraint at a time*, and the only communication between constraints is by deleting values.
- Automatically combining certain constraints is an active research topic.
    - But getting “the best possible” filtering from two “all different” constraints simultaneously is NP-hard. . .

This is Not The Exam Question

What is a *Hall set*, and why is it useful for propagation? Use the following model to illustrate your answer:

$$x_1 \in \{4,5\} \qquad x_2 \in \{1,2,3,4\} \qquad x_3 \in \{3,4,5\}$$
$$x_4 \in \{5,6\} \qquad x_5 \in \{3,5\}$$
$$\mathit{alldifferent}(x_1, x_2, x_3, x_4, x_5)$$

Suppose our solver did not have an “all different” constraint. Show how to rewrite this model using only binary constraints. What effect would this have on propagation?

Aside from propagation, describe another benefit of global constraints.

VIA VERITAS VITA

University of Glasgow