

Global Constraints and Consistency

Ciaran McCreech



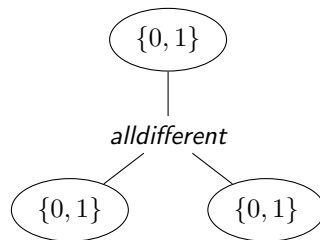
Two-Colouring a Triangle

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

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

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

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



Decomposing “All Different”

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

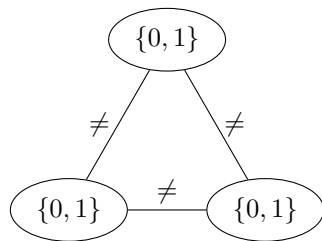
$$x_2 \in \{0, 1\}$$

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

$$x_1 \neq x_2$$

$$x_1 \neq x_3$$

$$x_2 \neq x_3$$



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.

What Does Propagation Really Do?

- For a not-equals constraint, no need to call it at all until one of the variables has only one value remaining.
- Initial propagation literally does nothing!

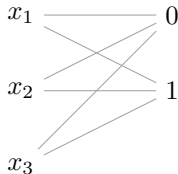
What Would a Human Do?

“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.

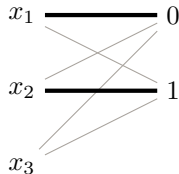
Revision: Matchings and All-Different

- 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.



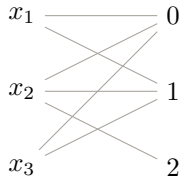
Revision: Matchings and All-Different

- 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.



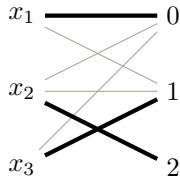
Revision: Matchings and All-Different

- 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.



Revision: Matchings and All-Different

- 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.



Sudoku

From Wikipedia, the free encyclopedia

Not to be confused with [Sudoku](#) or [Sudeki](#).

Sudoku (数独 *sūdoku*^[?], digit-single) ^[1]/suːˈdoʊkuː/, ^[2]/-ˈdɒ-/, ^[3]/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 A Constraint Solver Do?

If we work with a solver directly rather than using MiniZinc, we can propagate constraints and then stop, without searching. Here we'll use the Choco solver:

```
Model model = new Model();
IntVar x1 = model.intVar("x1", new int [] {1,8});
IntVar x2 = model.intVar("x2", new int [] {2,3});
IntVar x3 = model.intVar("x3", new int [] {2,3});
IntVar x4 = model.intVar("x4", new int [] {2,4,5});
IntVar x5 = model.intVar("x5", new int [] {4,5,6});
IntVar x6 = model.intVar("x6", new int [] {4,5,6});
IntVar x7 = model.intVar("x7", new int [] {2,7,9});
IntVar x8 = model.intVar("x8", new int [] {3,7,8});
IntVar x9 = model.intVar("x9", new int [] {2,3,5,8,9});
IntVar [] xs = new IntVar [] {x1,x2,x3,x4,x5,x6,x7,x8,x9};
model.allDifferent(xs).post();

System.out.println("Before:␣" + Arrays.toString(xs));
Solver solver = model.getSolver();
solver.propagate();
System.out.println("After:␣" + Arrays.toString(xs));
```

What Does A Constraint Solver Do?

Before:

```
[x1 = {1,8},    x2 = {2..3},    x3 = {2..3},  
 x4 = {2,4..5}, x5 = {4..6},    x6 = {4..6},  
 x7 = {2,7,9}, x8 = {3,7..8}, x9 = {2..3,5,8..9}]
```

After:

```
[x1 = 1,        x2 = {2..3},    x3 = {2..3},  
 x4 = {4..5},   x5 = {4..6},    x6 = {4..6},  
 x7 = {7,9},    x8 = {7..8},    x9 = {8..9}]
```

1	23	23	45	456	456	79	78	89
---	----	----	----	-----	-----	----	----	----

What Does A Constraint Solver Do?

What if we used not-equals instead?

```
for (int a = 0 ; a < xs.length ; ++a)
  for (int b = a + 1 ; b < xs.length ; ++b)
    model.arithm(xs[a], "!=" , xs[b]).post();
```

What Does A Constraint Solver Do?

Remember that not-equals does nothing unless a variable has only one value remaining.

Before:

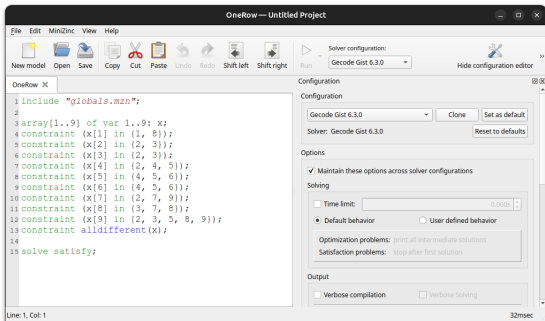
```
[x1 = {1,8},    x2 = {2..3},    x3 = {2..3},  
 x4 = {2,4..5}, x5 = {4..6},    x6 = {4..6},  
 x7 = {2,7,9},  x8 = {3,7..8},  x9 = {2..3,5,8..9}]
```

After:

```
[x1 = {1,8},    x2 = {2..3},    x3 = {2..3},  
 x4 = {2,4..5}, x5 = {4..6},    x6 = {4..6},  
 x7 = {2,7,9},  x8 = {3,7..8},  x9 = {2..3,5,8..9}]
```

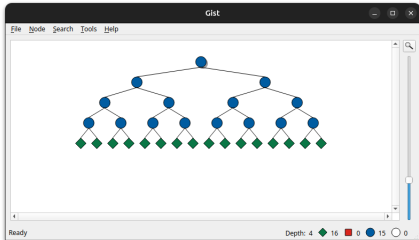
What Does A Constraint Solver Do?

We could also get this through the MiniZinc IDE by selecting the Gecode Gist solver and using the “Inspect” tool, although it’s a bit clumsy:



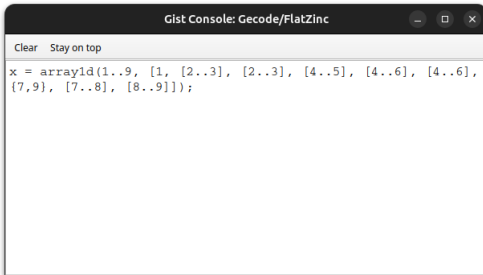
What Does A Constraint Solver Do?

We could also get this through the MiniZinc IDE by selecting the Gecode Gist solver and using the “Inspect” tool, although it’s a bit clumsy:



What Does A Constraint Solver Do?

We could also get this through the MiniZinc IDE by selecting the Gecode Gist solver and using the “Inspect” tool, although it’s a bit clumsy:

A screenshot of a window titled "Gist Console: Gecode/FlatZinc". The window has a title bar with standard window controls (minimize, maximize, close). Below the title bar is a toolbar with "Clear" and "Stay on top" buttons. The main area of the window contains a single line of MiniZinc code: `x = array1d(1..9, [1, [2..3], [2..3], [4..5], [4..6], [4..6], {7,9}, [7..8], [8..9]]);`

```
Gist Console: Gecode/FlatZinc
Clear Stay on top
x = array1d(1..9, [1, [2..3], [2..3], [4..5], [4..6], [4..6],
{7,9}, [7..8], [8..9]]);
```

Notice that Choco and Gecode do exactly the same thing. This is often not the case.

Generalised Arc Consistency

- Arc Consistency (AC): for a binary constraint, each value is supported by at least one value in the other variable.
- Generalised Arc Consistency (GAC): 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.
 - Each remaining value appears in at least one solution to the constraint.

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.

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.

GAC for All-Different

- 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! Here's a quick animation. For more, see one of the suggested poster papers.

GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------

GAC for All-Different

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]

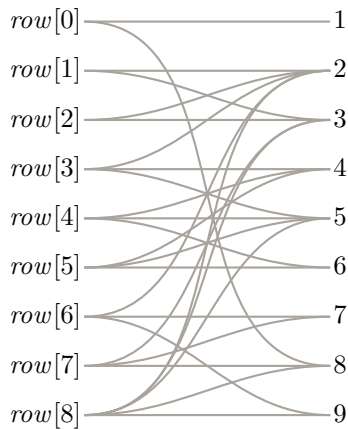
GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------

<i>row</i> [0]	1
<i>row</i> [1]	2
<i>row</i> [2]	3
<i>row</i> [3]	4
<i>row</i> [4]	5
<i>row</i> [5]	6
<i>row</i> [6]	7
<i>row</i> [7]	8
<i>row</i> [8]	9

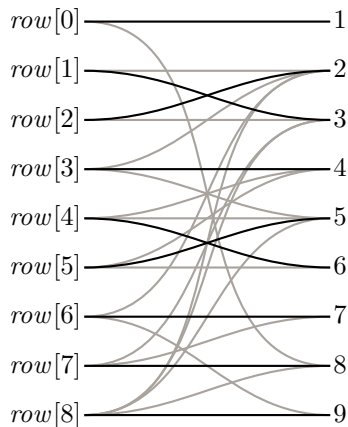
GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------



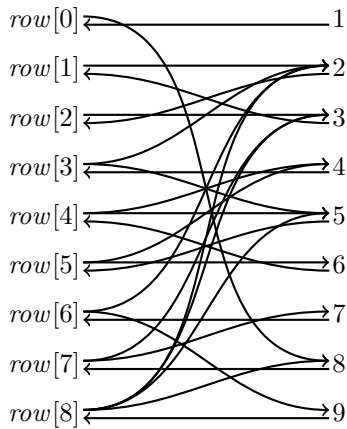
GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------



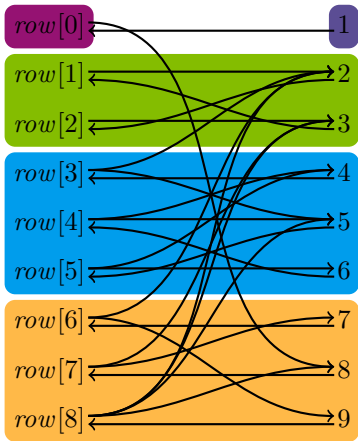
GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------



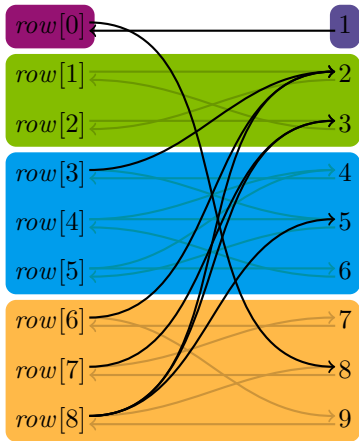
GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------



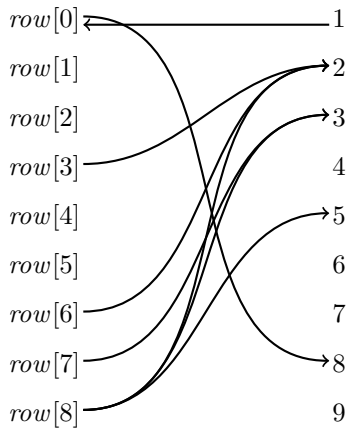
GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------



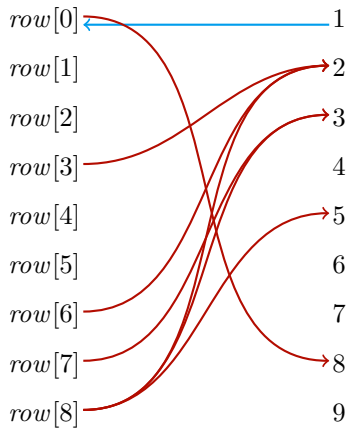
GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------



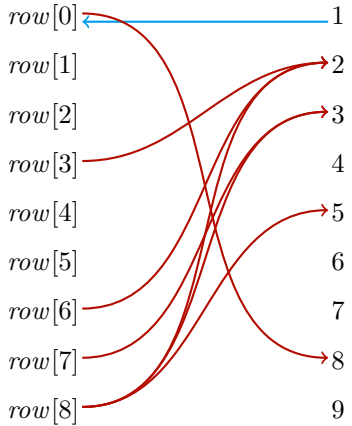
GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------




GAC for All-Different

18	23	23	245	456	456	279	378	23589
----	----	----	-----	-----	-----	-----	-----	-------




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();
}

Model model = new Model();
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();

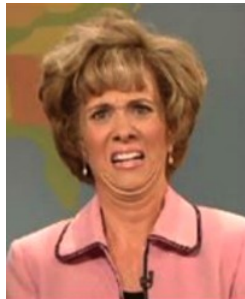
// Columns
for (int i = 0 ; i < nn ; ++i) {
    IntVar[] column = new IntVar[nn];
    for (int j = 0 ; j < nn ; ++j)
        column[j] = grid[j][i];
    model.allDifferent(column).post();
}
```

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

```
Solver solver = model.getSolver();
if (solver.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" + solver.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

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.033s
  Resolution time : 0.019s
  Nodes: 1 (52.6 n/s)
```

Some Experiments

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.033s
  Resolution time : 0.019s
  Nodes: 1 (52.6 n/s)
```

Using all-different:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.034s
  Resolution time : 0.020s
  Nodes: 1 (50.4 n/s)
```

Some Experiments

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.033s
  Resolution time : 0.019s
  Nodes: 1 (52.6 n/s)
```

Using all-different:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.034s
  Resolution time : 0.020s
  Nodes: 1 (50.4 n/s)
```

Both solved without guessing (Nodes: 1). Using not equals is 1ms faster to set up and 1ms faster to solve. We could run it lots of times to test for statistical significance, but do we care?

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

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.034s
  Resolution time : 0.029s
  Nodes: 20 (688.6 n/s)
```

Some Experiments

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.034s
  Resolution time : 0.029s
  Nodes: 20 (688.6 n/s)
```

Using all-different:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.042s
  Resolution time : 0.031s
  Nodes: 1 (32.7 n/s)
```


Some Experiments

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.034s
  Resolution time : 0.029s
  Nodes: 20 (688.6 n/s)
```

Using all-different:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.042s
  Resolution time : 0.031s
  Nodes: 1 (32.7 n/s)
```

All-different can solve without guessing (Nodes: 1 vs 20) but is 2ms slower to run and 8ms slower to set up.

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
```

- Complete search - 1 solution found.

Solutions: 1

Building time : 0.032s

Resolution time : 0.032s

Nodes: 30 (950.7 n/s)

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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.032s
  Resolution time : 0.032s
  Nodes: 30 (950.7 n/s)
```

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.035s
  Resolution time : 0.024s
  Nodes: 2 (83.6 n/s)
```

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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.032s
  Resolution time : 0.032s
  Nodes: 30 (950.7 n/s)
```

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.035s
  Resolution time : 0.024s
  Nodes: 2 (83.6 n/s)
```

Less guessing from all-different again, and it's slightly faster.

Some Experiments

```
0 3 0 9 7 5 0 1 0
1 0 0 4 0 2 0 0 5
6 0 0 0 0 0 0 0 2
0 6 0 0 0 0 0 2 0
9 0 0 3 0 6 0 0 1
0 1 0 0 0 0 0 6 0
8 0 0 0 0 0 0 0 4
4 0 0 2 0 1 0 0 8
0 2 0 8 9 4 0 5 0
//
// Times 15/10/2007
// super fiendish
//
```

Some Experiments

Using not-equals:

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

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.032s
  Resolution time : 0.026s
  Nodes: 12 (462.2 n/s)
```

Some Experiments

Using not-equals:

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

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.032s
  Resolution time : 0.026s
  Nodes: 12 (462.2 n/s)
```

Using all-different:

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

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.035s
  Resolution time : 0.028s
  Nodes: 3 (108.9 n/s)
```


Some Experiments

Using not-equals:

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

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.032s
  Resolution time : 0.026s
  Nodes: 12 (462.2 n/s)
```

Using all-different:

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

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.035s
  Resolution time : 0.028s
  Nodes: 3 (108.9 n/s)
```

Things aren't looking good for all-different...

Some Experiments

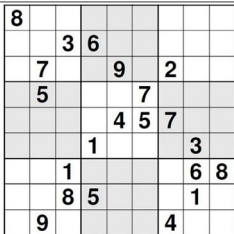
The Telegraph



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.



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 »
Arto Inkala

Some Experiments

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.033s
  Resolution time : 0.126s
  Nodes: 884 (7,010.2 n/s)
```

Some Experiments

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.033s
  Resolution time : 0.126s
  Nodes: 884 (7,010.2 n/s)
```

Using all-different:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.035s
  Resolution time : 0.077s
  Nodes: 89 (1,160.8 n/s)
```

Some Experiments

Using not-equals:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.033s
  Resolution time : 0.126s
  Nodes: 884 (7,010.2 n/s)
```

Using all-different:

```
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
```

```
- Complete search - 1 solution found.
  Solutions: 1
  Building time : 0.035s
  Resolution time : 0.077s
  Nodes: 89 (1,160.8 n/s)
```

Finally, an epic and extremely convincing victory for all-different, we saved an entire 47ms overall.

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 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 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 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 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 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 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 31 0 3 0 6 9
30 29 0 12 0 5 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 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 9 0 0 0
0 0 17 25 20 23 0 30 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 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 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 6 0 0 4 0 0 32 0 0 20
0 0 7 3 0 0 22 0 0 13 0 0 21 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 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 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 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

Using not-equals:

Don't know, I gave up after 2 hours.

Some Experiments

Using not-equals:

Don't know, I gave up after 2 hours.

Using all-different:

```
- Complete search - 1 solution found.  
  Solutions: 1  
  Building time : 0.067s  
  Resolution time : 0.115s  
  Nodes: 1 (8.7 n/s)
```


Some Experiments

Using not-equals:

```
Don't know, I gave up after 2 hours.
```

Using all-different:

```
- Complete search - 1 solution found.  
  Solutions: 1  
  Building time : 0.067s  
  Resolution time : 0.115s  
  Nodes: 1 (8.7 n/s)
```

Given Nodes: 1, nothing clever is needed to solve this, so a really bored human could probably do this faster than the not-equals version.

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 only have weaker propagators: GAC can be too hard to be practical, or NP-complete on its own.
- It is impossible to get GAC for all-different from a polynomial-sized SAT encoding.
- 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.

Consistency, Again

- Consistency depends upon how exactly your constraints are specified.
- GAC is the strongest possible consistency level, if all we can do is consider one constraint at once and delete values from domains.

Are You Smarter than a Constraint Solver?

		45						
	34	35		345				

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...

Not the Exam Question Because There is No Exam

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

$$x_1 \in \{4, 5\}$$

$$x_2 \in \{1, 2, 3, 4\}$$

$$x_3 \in \{3, 4, 5\}$$

$$x_4 \in \{5, 6\}$$

$$x_5 \in \{3, 5\}$$

$$\text{alldifferent}(x_1, x_2, x_3, x_4, x_5)$$

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.

Linear (or Scalar) Products

- Let c be an array of constants and v an array of variables. The *scalar product* of c and v is

$$c \cdot v = \sum_{i \in A} c[i] \times v[i]$$

- We constrain this with an operator to another variable (or constant). The operator can be an equation ($=$), or an inequality (\leq or \geq).
- In MiniZinc, the compiler automatically detects constraints like $3 * x + 2 * y \leq 10$ and $sum(i \text{ in } s)(a[i] * b[i]) = 12$.
- These are known as *linear* equalities and inequalities, because geometrically they define a straight line.

What Can We Do?

$$x_1 \in \{0, 2\}$$

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

$$x_3 \in \{0, 2\}$$

$$5x_1 + 2x_2 + x_3 \geq 8$$

What Can We Do?

$$x_1 \in \{0, 2\}$$

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

$$x_3 \in \{0, 2\}$$

$$5x_1 + 2x_2 + x_3 \geq 8$$

- x_1 must be at least 1, because we can get at most $(2 * 1) + (1 * 2) = 4$ from the other two variables.

What Can We Do?

$$x_1 \in \{0, 2\}$$

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

$$x_3 \in \{0, 2\}$$

$$5x_1 + 2x_2 + x_3 \geq 8$$

- x_1 must be at least 1, because we can get at most $(2 * 1) + (1 * 2) = 4$ from the other two variables.
- x_2 and x_3 don't change, because $x_1 = 2$ can satisfy the inequality on its own.

What Can We Do?

$$x_1 \in \{0, 2\}$$

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

$$x_3 \in \{0, 2\}$$

$$5x_1 + 2x_2 + x_3 \geq 8$$

- x_1 must be at least 1, because we can get at most $(2 * 1) + (1 * 2) = 4$ from the other two variables.
- x_2 and x_3 don't change, because $x_1 = 2$ can satisfy the inequality on its own.
- What if $x_1 \in \{0, 1\}$?

What Can We Do?

$$x_1 \in \{0, 2\}$$

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

$$x_3 \in \{0, 2\}$$

$$5x_1 + 2x_2 + x_3 \geq 8$$

- x_1 must be at least 1, because we can get at most $(2 * 1) + (1 * 2) = 4$ from the other two variables.
- x_2 and x_3 don't change, because $x_1 = 2$ can satisfy the inequality on its own.
- What if $x_1 \in \{0, 1\}$?
 - x_2 can't be 0, because we can only get $(5 * 1) + (1 * 2) = 7$ from the other two without its help.

What Can We Do?

$$x_1 \in \{0, 2\}$$

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

$$x_3 \in \{0, 2\}$$

$$5x_1 + 2x_2 + x_3 \geq 8$$

- x_1 must be at least 1, because we can get at most $(2 * 1) + (1 * 2) = 4$ from the other two variables.
- x_2 and x_3 don't change, because $x_1 = 2$ can satisfy the inequality on its own.
- What if $x_1 \in \{0, 1\}$?
 - x_2 can't be 0, because we can only get $(5 * 1) + (1 * 2) = 7$ from the other two without its help.
 - x_3 can't be 0, because we can only get $(5 * 1) + (2 * 1) = 7$ from the other two variables without its help.

The Slack Algorithm for \geq Inequalities

- Feasibility: if every variable contributes as much as possible, do we have enough?
- Filtering: for each variable in turn, what's the most the remaining variables can contribute, and how much do I have to contribute if this happens?
 - Assuming only positive numbers, this might increase lower bounds.
- Actual algorithm removed for mental health reasons. . .

When Do Linear Inequalities Propagate?

- For positive coefficients and \geq : only when an upper bound changes.
- Can be even more specific sometimes...
 - For example, if one variable gets a very large value, maybe the constraint will always be satisfied.

In Choco

```
Model model = new Model("Scalar");
Solver solver = model.getSolver();

IntVar x1 = model.intVar("x1", 2, 5);
IntVar x2 = model.intVar("x2", 1, 50);
IntVar x3 = model.intVar("x3", 0, 2);

model.scalar(
    new IntVar[] { x1, x2, x3 },
    new int[] { 2, -2, 4 },
    ">=", 11).post();

System.out.println(model);

solver.propagate();

System.out.println(model);
```


In Choco

```
Model[Scalar]
```

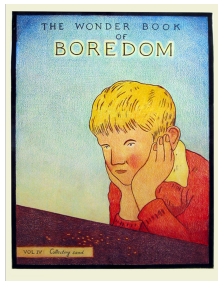
```
[ 4 vars -- 1 cstrs ]  
satisfaction : undefined  
== variables ==  
x1 = {2..5}  
x2 = {1..50}  
x3 = {0..2}  
cste -- 11 = 11  
== constraints ==  
SUM ([2.x1 + 4.x3 - 2.x2 >= 11])
```

```
Model[Scalar]
```

```
[ 4 vars -- 1 cstrs ]  
satisfaction : undefined  
== variables ==  
x1 = {3..5}  
x2 = {1..3}  
x3 = {1..2}  
cste -- 11 = 11  
== constraints ==  
SUM ([2.x1 + 4.x3 - 2.x2 >= 11])
```

Remember Generalised Arc Consistency?

- A *binary* constraint is arc consistent if we can pick any value from either of its two variables, and find a supporting value in the other variable.
- A global constraint is generalised arc consistent (GAC) if each value is present in at least one solution to the constraint.
- Enforcing GAC will only touch upper and lower bounds for linear inequalities.
 - We could also define *bounds* consistency.



Linear Equalities

- What if we have an equation, rather than an inequality?

$$x = \sum_{i \in A} c[i] \times v[i]$$

Linear Equalities

- What if we have an equation, rather than an inequality?

$$x = \sum_{i \in A} c[i] \times v[i]$$

- The subset sum problem is NP-complete...

Linear Equalities

- What if we have an equation, rather than an inequality?

$$x = \sum_{i \in A} c[i] \times v[i]$$

- The subset sum problem is NP-complete...
- We can enforce GAC on a \leq constraint and a \geq constraint simultaneously. This is **not** the same as enforcing GAC on the equality.

In Choco

```
Model model = new Model("ScalarEquality");
Solver solver = model.getSolver();

IntVar x1 = model.intVar("x1", 2, 5);
IntVar x2 = model.intVar("x2", 2, 5);
IntVar x3 = model.intVar("x3", 2, 5);

model.scalar(
    new IntVar[] { x1, x2, x3 },
    new int[] { 2, -2, 4 },
    "=", 11).post();

System.out.println(model);

solver.propagate();

System.out.println(model);
```

In Choco

```
Model[ScalarEquality]

[ 4 vars -- 1 cstrs ]
satisfaction : undefined
== variables ==
x1 = {2..5}
x2 = {2..5}
x3 = {2..5}
cste -- 11 = 11
== constraints ==
TABLE ([CSPLarge({x3 = {2..5}, , x2 = {2..5}, , x1 = {2..5}, })])

Exception in thread "main" CONTRADICTION (CSPLarge(
  {x3 = {2..5}, , x2 = {2..5}, , x1 = {2..5}, },
  x3 = {2..5}) : new upper bound is lesser than lower bound
```

In Choco

```
Model[ScalarEquality]

[ 4 vars -- 1 cstrs ]
satisfaction : undefined
== variables ==
x1 = {2..5}
x2 = {2..5}
x3 = {2..5}
cste -- 11 = 11
== constraints ==
TABLE ([CSPLarge({x3 = {2..5}, , x2 = {2..5}, , x1 = {2..5}, })]])
```

```
Exception in thread "main" CONTRADICTION (CSPLarge(
  {x3 = {2..5}, , x2 = {2..5}, , x1 = {2..5}, },
  x3 = {2..5}) : new upper bound is lesser than lower bound
```

Somehow it's figured out the constraint is infeasible, and it says our constraint is a Table, not a Sum.

In Choco (Attempt Two)

Let's try some bigger numbers and see what happens.

```
Model model = new Model("ScalarEquality");
Solver solver = model.getSolver();

IntVar x1 = model.intVar("x1", new int[] { 0, 2, 10 });
IntVar x2 = model.intVar("x2", new int[] { 0, 2, 10 });
IntVar x3 = model.intVar("x3", new int[] { 0, 2, 10 });
IntVar x4 = model.intVar("x4", new int[] { 0, 2, 10 });

model.scalar(
    new IntVar[] { x1, x2, x3, x4 },
    new int[] { 1, 1, 1, 1 },
    "=", 5).post();

System.out.println(model);

solver.propagate();

System.out.println(model);
```

In Choco (Attempt Two)

```
Model[ScalarEquality]
```

```
[ 5 vars -- 1 cstrs ]  
satisfaction : undefined  
== variables ==  
x1 = {0,2,10}  
x2 = {0,2,10}  
x3 = {0,2,10}  
x4 = {0,2,10}  
cste -- 5 = 5  
== constraints ==  
SUM ([x1 + x3 + x2 + x4 = 5])
```

```
Model[ScalarEquality]
```

```
[ 5 vars -- 1 cstrs ]  
satisfaction : undefined  
== variables ==  
x1 = {0,2}  
x2 = {0,2}  
x3 = {0,2}  
x4 = {0,2}  
cste -- 5 = 5  
== constraints ==  
SUM ([x1 + x3 + x2 + x4 = 5])
```

When Do Linear Equalities Propagate?

- As a pair of inequalities, when bounds change.
- For GAC, it's complicated. . .

I'm Telling You This Twice Because It's Important

- Propagation and consistency both depend upon how exactly you represent constraints.
- Modelling languages and solvers will *sometimes* help you out.

Jess Probably Isn't Mean Enough To Ask This On A Test

Consider the following constraint:

$$2x_1 - 2x_2 + 4x_3 \geq 11$$

Given $x_1 \in \{2, 3, 4, 5\}$, $x_2 \in \{1, 2, 3, 50\}$ and $x_4 \in \{0, 1, 2\}$, prove $x_2 \neq 50$. What else can we infer?

Now suppose we add a second constraint,

$$2x_1 - 2x_2 + 4x_3 \leq 11$$

Why is enforcing consistency on these two constraints *not* the same as enforcing consistency on this single equality constraint?

$$2x_1 - 2x_2 + 4x_3 = 11$$

Table Constraints

- We can represent a constraint as a list of permitted tuples:

$$[A, B, C] \in \{[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]\}$$

- This is the preferred definition for theoretical computer science, but not commonly used in practice.
- Sometimes table constraints *are* the best option, though.

Consistency, Round Three

- Back to generalised arc consistency!
- Call a tuple *feasible* if each of its values remains in the domain of its respective variable.
- If each value in each variable is present in at least one feasible tuple, we have GAC.

Propagating Table Constraints

- Keep track of remaining feasible tuples.
- For example, by using an additional “variable”,

$$T = 1 \rightarrow (A = 1 \wedge B = 2 \wedge C = 3)$$

$$T = 2 \rightarrow (A = 1 \wedge B = 1 \wedge C = 2)$$

$$T = 3 \rightarrow (A = 2 \wedge B = 1 \wedge C = 3)$$

$$T = 4 \rightarrow (A = 2 \wedge B = 3 \wedge C = 1)$$

$$T = 5 \rightarrow (A = 3 \wedge B = 1 \wedge C = 2)$$

$$T = 6 \rightarrow (A = 3 \wedge B = 2 \wedge C = 1)$$

- Remove any value that is not supported in a feasible tuple.

Why Not Just Use Table Constraints?

- We can propagate table constraints in (low) polynomial time, and get GAC.
 - There are much faster algorithms than the one on the previous slide.
 - See: one of the suggested poster papers.
- Why not use table constraints for everything?

Why Not Just Use Table Constraints?

- We can propagate table constraints in (low) polynomial time, and get GAC.
 - There are much faster algorithms than the one on the previous slide.
 - See: one of the suggested poster papers.
- Why not use table constraints for everything?
- Tables can be exponentially long, e.g. for all-different.
 - Would it be possible to redo the Sudoku experiments using table constraints?

Autotabulation

- Remember Choco being clever with linear equalities?
- A solver can turn one or more constraints into a table constraint to get more powerful propagation.
- If you know when to do this, it can be really powerful.
- See: one of the suggested poster papers.

Beyond Table Constraints

- Smart tables: wildcards, negations, and ranges.
 - Can make tables much smaller for some problems.
- Decision diagrams.
 - Even more compact representations.
- Regular expressions.
 - See: one of the suggested poster papers.

Shift Pattern Scheduling

- Produce a schedule for eleven employees for two weeks.
- Each employee can be on day shift, night shift, or off.
- Four employees present on each day shift.
- Two employees present on each night shift.
- Can't work more than five day shifts in a row.
- Can't work more than two night shifts in a row.
- Day shift can only be followed by a day shift or a day off.
- Night shift can only be followed by a night shift or a day off.

Shift Pattern Scheduling

```
include "globals.mzn";

enum Shifts = { Day, Night, Off };
enum Days = anon_enum(14);
enum Workers = anon_enum(11);

int: day_cover = 4;
int: night_cover = 2;

array[Days, Workers] of var Shifts: allocation;

constraint forall (d in Days)
  (sum(w in Workers)(allocation[d, w] == Day) = day_cover);
constraint forall (d in Days)
  (sum(w in Workers)(allocation[d, w] == Night) = night_cover);

constraint forall (w in Workers)(regular([allocation[d, w] | d in Days],
  "(Off|(Day{1,5} Off)|(Night{1,2} Off))* (Day{1,5}|Night{1,2}|Off)"));

solve ::int_search([allocation[d, w] | d in Days, w in Workers],
  first_fail, indomain_min) satisfy;

output [ join(" ", [
  format_justify_string(-5, show(allocation[d, w])) | d in Days
]) ++ "\n" | w in Workers];
```

Shift Pattern Scheduling

Running yaycapitalism1.mzn

```
Day Day Day Day Day Off Night Night Off Night Night Off Day Day
Day Day Day Day Day Off Night Night Off Night Night Off Night Night
Day Day Day Day Day Off Off Day Day Day Day Day Day Off Off
Day Day Day Day Day Off Off Off Night Off Day Day Day Day
Night Night Off Night Night Off Off Off Night Off Day Day Day Day
Night Night Off Night Off Day Day Day Day Day Off Night Off Off
Off Off Night Off Night Night Night Off Off Off Off Day Day Day
Off Off Night Off Off Day Day Day Day Day Off Night Off Off
Off Off Off Off Off Day Day Day Day Day Off Off Night Night
Off Off Off Off Off Day Day Off Off Off Off Off Off Off Off
```

Finished in 178msec.

Shift Pattern Scheduling

```
Running yaycapitalism1.mzn
```

```
Day Day Day Day Day Off Night Night Off Night Night Off Day Day
Day Day Day Day Day Off Night Night Off Night Night Off Night Night
Day Day Day Day Day Off Off Day Day Day Day Day Day Off Off
Day Day Day Day Day Off Off Off Night Off Day Day Day Day
Night Night Off Night Night Off Off Off Night Off Day Day Day Day
Night Night Off Night Off Day Day Day Day Day Off Night Off Off
Off Off Night Off Night Night Off Off Off Off Off Day Day Day
Off Off Night Off Off Day Day Day Day Day Off Night Off Off
Off Off Off Off Off Day Day Day Day Day Off Off Night Night
Off Off Off Off Off Day Day Off Off Off Off Off Off Off
Off Off Off Off Off Night Off Off Off Off Off Off Off Off
```

```
-----
Finished in 178msec.
```

Those last two employees might not be very happy...

Shift Pattern Scheduling

```
include "globals.mzn";

enum Shifts = { Day, Night, Off };
enum Days = anon_enum(14);
enum Workers = anon_enum(11);

int: day_cover = 4;
int: night_cover = 2;

array[Days, Workers] of var Shifts: allocation;

constraint forall (d in Days)
  (sum(w in Workers)(allocation[d, w] == Day) = day_cover);
constraint forall (d in Days)
  (sum(w in Workers)(allocation[d, w] == Night) = night_cover);

constraint forall (w in Workers)(regular([allocation[d, w] | d in Days],
  "(Off|(Day{1,5} Off)|(Night{1,2} Off))* (Day{1,5}|Night{1,2}|Off)"));

var 0..card(Workers): layoffs;
constraint forall (w in Workers, d in Days)(layoffs >= w -> allocation[d, w] = Off);

solve ::int_search([allocation[d, w] | d in Days, w in Workers],
  first_fail, indomain_min) maximize layoffs;

output [ join(" ", [
  format_justify_string(-5, show(allocation[d, w])) | d in Days
]) ++ "\n" | w in Workers];
```

Shift Pattern Scheduling

```
Running yaycapitalism2.mzn
```

```
-----  
Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  
Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  
Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  Off  
Day  Day  Day  Day  Day  Off  Night Night Off  Day  Day  Day  Day  
Day  Day  Day  Day  Day  Off  Night Night Off  Day  Day  Day  Day  
Day  Day  Day  Off  Night Night Off  Day  Day  Day  Day  Off  Night Night  
Day  Day  Day  Off  Night Night Off  Day  Day  Day  Day  Off  Night Night  
Night Night Off  Day  Day  Day  Day  Off  Night Night Off  Day  Day  Day  
Night Night Off  Day  Day  Day  Day  Off  Night Night Off  Day  Day  Day  
Off  Off  Night Night Off  Day  Day  Day  Day  Off  Night Night Off  Off  
Off  Off  Night Night Off  Day  Day  Day  Day  Off  Night Night Off  Off  
-----  
=====
```

```
Finished in 3s 79msec.
```

Shift Pattern Scheduling

- You've seen something similar using implications rather than regular.
- Given another hour or two, I could prove to you that GAC on the regular constraint can do more filtering in some situations.
- Doesn't necessarily mean it runs any faster.
- Whether or not it is more readable depends upon how you feel about regular expressions. . .

Ethical Considerations

- Easy to write a model that produces a valid schedule.
- Hard to write a *good* model that doesn't ruin people's lives.
 - *Balance* constraints for minimum and maximum hours.
 - Optimise to respect employee preferences.

Ethical Considerations

- Easy to write a model that produces a valid schedule.
- Hard to write a *good* model that doesn't ruin people's lives.
 - *Balance* constraints for minimum and maximum hours.
 - Optimise to respect employee preferences.
- Ethical obligation to support our democratically elected capitalist government by maximising shareholder value:
 - Give fewest hours to most expensive (or least productive) employees?
 - No need to lay anyone off, just use zero hour contracts.
 - Get rid of poorly performing employees by giving them bad shift patterns.

