Global Constraints

- A *global constraint* is one which can operate on arbitrarily many variables.
Two-Colouring a Triangle

\[ x_1 \in \{0, 1\} \]
\[ x_2 \in \{0, 1\} \]
\[ x_3 \in \{0, 1\} \]

\textit{alldifferent}(x_1, x_2, x_3)

\{0, 1\}
Decomposing “All Different”

\[ x_1 \in \{0, 1\} \]
\[ x_2 \in \{0, 1\} \]
\[ x_3 \in \{0, 1\} \]
\[ x_1 \neq x_2 \]
\[ x_1 \neq x_3 \]
\[ x_2 \neq x_3 \]

\[
\begin{array}{cccc}
\{0, 1\} & \neq & \{0, 1\} \\
\{0, 1\} & \neq & \{0, 1\} \\
\{0, 1\} & \neq & \{0, 1\}
\end{array}
\]
What Does Propagation Do?

- Let’s consider the constraint $x_1 \neq x_2$.
  - 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 su dōku, digit-single) /suːˈdoʊkuː/, /-ˈdoʊ-, /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.

A typical Sudoku puzzle

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>1</td>
<td>9</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>9</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

The same puzzle with solution numbers marked in red

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>
How do Humans Solve Sudoku?

<table>
<thead>
<tr>
<th>18</th>
<th>23</th>
<th>23</th>
<th>245</th>
<th>456</th>
<th>456</th>
<th>279</th>
<th>378</th>
<th>23589</th>
</tr>
</thead>
</table>

Ciaran McCreesh and Patrick Prosser
The “All Different” Constraint
How do Humans Solve Sudoku?

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>23</td>
<td>23</td>
<td>245</td>
<td>456</td>
<td>456</td>
<td>279</td>
<td>378</td>
<td>23589</td>
</tr>
</tbody>
</table>
### How do Humans Solve Sudoku?

<table>
<thead>
<tr>
<th></th>
<th>23</th>
<th>23</th>
<th>245</th>
<th>456</th>
<th>456</th>
<th>279</th>
<th>378</th>
<th>23589</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How do Humans Solve Sudoku?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>23</th>
<th>23</th>
<th>245</th>
<th>456</th>
<th>456</th>
<th>279</th>
<th>378</th>
<th>23589</th>
</tr>
</thead>
</table>

Ciaran McCreesh and Patrick Prosser
The “All Different” Constraint
How do Humans Solve Sudoku?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>23</th>
<th>23</th>
<th>245</th>
<th>456</th>
<th>456</th>
<th>279</th>
<th>378</th>
<th>23589</th>
</tr>
</thead>
</table>

Ciaran McCreesh and Patrick Prosser
How do Humans Solve Sudoku?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>23</th>
<th>23</th>
<th>245</th>
<th>456</th>
<th>456</th>
<th>279</th>
<th>378</th>
<th>23589</th>
</tr>
</thead>
</table>

Ciaran McCreesh and Patrick Prosser

The “All Different” Constraint
How do Humans Solve Sudoku?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>23</th>
<th>23</th>
<th>45</th>
<th>456</th>
<th>456</th>
<th>79</th>
<th>78</th>
<th>589</th>
</tr>
</thead>
</table>
How do Humans Solve Sudoku?

1 23 23 45 456 456 79 78 589
How do Humans Solve Sudoku?

\[
\begin{array}{cccccccc}
1 & 23 & 23 & 45 & 456 & 456 & 79 & 78 & 589 \\
\end{array}
\]
How do Humans Solve Sudoku?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>23</th>
<th>23</th>
<th>45</th>
<th>456</th>
<th>456</th>
<th>79</th>
<th>78</th>
<th>89</th>
</tr>
</thead>
</table>
What Does Choco Do?

Solver solver = new Solver("one_row_sudoku");
IntVar[] row = enumeratedArray("row", 9, 1, 9, solver);

solver.post(member(row[0], new int[] { 1, 8 }));
solver.post(member(row[1], new int[] { 2, 3 }));
solver.post(member(row[2], new int[] { 2, 3 }));
solver.post(member(row[3], new int[] { 2, 4, 5 }));
solver.post(member(row[4], new int[] { 4, 5, 6 }));
solver.post(member(row[5], new int[] { 4, 5, 6 }));
solver.post(member(row[6], new int[] { 2, 7, 9 }));
solver.post(member(row[7], new int[] { 3, 7, 8 }));
solver.post(member(row[8], new int[] { 2, 3, 5, 8, 9 }));

solver.propagate();
System.out.println("Before: " + Arrays.toString(row));

solver.post(allDifferent(row));
solver.propagate();
System.out.println("After: " + Arrays.toString(row));
What Does Choco Do?

Before:

| row[0] = {1,8}, | row[1] = {2,3}, | row[2] = {2,3}, |

After:


| 1  | 23 | 23 | 45 | 456 | 456 | 79 | 78 | 89 |
What Does Choco Do?

// solver.post(alldifferent(row));

for (int i = 0 ; i < 8 ; ++i)
    for (int j = i + 1 ; j < 9 ; ++j)
        solver.post(arithm(row[i], "!=", row[j]));
What Does Choco Do?

<table>
<thead>
<tr>
<th>Row</th>
<th>Before</th>
<th>After with all-different</th>
<th>After with not-equals</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{1,8}</td>
<td>{1}</td>
<td>{1,8}</td>
</tr>
<tr>
<td>1</td>
<td>{2,3}</td>
<td>{2,3}</td>
<td>{2,3}</td>
</tr>
<tr>
<td>2</td>
<td>{2,3}</td>
<td>{2,3}</td>
<td>{2,3}</td>
</tr>
<tr>
<td>3</td>
<td>{2,4,5}</td>
<td>{4,5}</td>
<td>{2,4,5}</td>
</tr>
<tr>
<td>4</td>
<td>{4,5,6}</td>
<td>{4,5,6}</td>
<td>{4,5,6}</td>
</tr>
<tr>
<td>5</td>
<td>{4,5,6}</td>
<td>{4,5,6}</td>
<td>{4,5,6}</td>
</tr>
<tr>
<td>6</td>
<td>{2,7,9}</td>
<td>{7,9}</td>
<td>{2,7,9}</td>
</tr>
<tr>
<td>7</td>
<td>{3,7,8}</td>
<td>{7,8}</td>
<td>{3,7,8}</td>
</tr>
<tr>
<td>8</td>
<td>{2,3,5,8,9}</td>
<td>{8,9}</td>
<td>{2,3,5,8,9}</td>
</tr>
</tbody>
</table>
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.
“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?

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>23</td>
<td>23</td>
<td>245</td>
<td>456</td>
<td>456</td>
<td>279</td>
<td>378</td>
<td>23589</td>
<td></td>
</tr>
</tbody>
</table>
Is Choco Magic?

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>23</td>
<td>23</td>
<td>245</td>
<td>456</td>
<td>456</td>
<td>279</td>
<td>378</td>
<td>23589</td>
</tr>
</tbody>
</table>

`row[0]`

`row[1]`

`row[2]`

`row[3]`

`row[4]`

`row[5]`

`row[6]`

`row[7]`

`row[8]`
Is Choco Magic?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

| 18 | 23 | 23 | 245 | 456 | 456 | 279 | 378 | 23589 |
Is Choco Magic?

\[
\begin{align*}
row[0] & \rightarrow 1 \\
row[1] & \rightarrow 2 \\
row[2] & \rightarrow 3 \\
row[3] & \rightarrow 4 \\
row[4] & \rightarrow 5 \\
row[5] & \rightarrow 6 \\
row[6] & \rightarrow 7 \\
row[7] & \rightarrow 8 \\
row[8] & \rightarrow 9
\end{align*}
\]
Is Choco Magic?

row[0] ——— 1
row[1] ——— 2
row[2] ——— 3
row[3] ——— 4
row[6] ——— 7
row[7] ——— 8
row[8] ——— 9
Is Choco Magic?

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?

```
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?

```
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?

```
row[0] -> 1
row[1] -> 2
row[2] -> 3
row[3] -> 4
row[5] -> 6
row[6] -> 7
row[7] -> 8
row[8] -> 9
```
Is Choco Magic?

```
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?
Is Choco Magic?

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

```java
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

Solver solver = new Solver("sudoku");

IntVar[][] grid = enumeratedMatrix("grid", nn, nn,
    IntStream.rangeClosed(1, nn).toArray(), solver);
A Sudoku Solver in Choco

// Rows
for (int i = 0 ; i < nn ; ++i)
    solver.post(alldifferent(grid[i]));
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];
    solver.post(alldifferent(column));
}
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];
        solver.post(alldifferent(square));
    }
}
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];
        solver.post(allDifferent(square));
    }
}
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])
      solver.post(arithm(grid[i][j], "=" , predef[i][j]));
A Sudoku Solver in Choco

System.out.println(solver.findSolution());
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());
Incidentally, 2D arrays in Gecode are Much Nicer

Matrix<IntVarArray> grid(x, nn, nn);

// Rows and columns
for (int i = 0; i < nn; i++) {
    distinct(*this, grid.row(i));
    distinct(*this, grid.col(i));
}

// Squares
for (int i = 0; i < nn; i += n)
    for (int j = 0; j < nn; j += n)
        distinct(*this, grid.slice(i, i + n, j, j + n));

// Fill-in predefined fields
for (int i = 0; i < nn; i++)
    for (int j = 0; j < nn; j++)
        if (0 != predef[i][j])
            rel(*this, grid(i, j) == predef[i][j]);
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

Solutions: 1
Initial propagation : 0.018s
Resolution : 0.020s
Nodes: 1 (50.2 n/s)
## Some Experiments

<table>
<thead>
<tr>
<th>Using not-equals:</th>
<th>Using all-different:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 6 3 1 2 7 4 9 5</td>
<td>8 6 3 1 2 7 4 9 5</td>
</tr>
<tr>
<td>1 9 2 5 4 3 6 8 7</td>
<td>1 9 2 5 4 3 6 8 7</td>
</tr>
<tr>
<td>7 5 4 9 6 8 2 1 3</td>
<td>7 5 4 9 6 8 2 1 3</td>
</tr>
<tr>
<td>9 3 1 8 5 6 7 4 2</td>
<td>9 3 1 8 5 6 7 4 2</td>
</tr>
<tr>
<td>6 2 5 4 7 1 8 3 9</td>
<td>6 2 5 4 7 1 8 3 9</td>
</tr>
<tr>
<td>4 8 7 2 3 9 5 6 1</td>
<td>4 8 7 2 3 9 5 6 1</td>
</tr>
<tr>
<td>3 4 6 7 1 2 9 5 8</td>
<td>3 4 6 7 1 2 9 5 8</td>
</tr>
<tr>
<td>5 7 9 3 8 4 1 2 6</td>
<td>5 7 9 3 8 4 1 2 6</td>
</tr>
<tr>
<td>2 1 8 6 9 5 3 7 4</td>
<td>2 1 8 6 9 5 3 7 4</td>
</tr>
</tbody>
</table>

### Solutions:
- Using not-equals: 1 solution
  - Initial propagation: 0.018s
  - Resolution: 0.020s
  - Nodes: 1 (50.2 n/s)

- Using all-different: 1 solution
  - Initial propagation: 0.020s
  - Resolution: 0.022s
  - Nodes: 1 (45.6 n/s)
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

Solutions: 1
Initial propagation : 0.018s
Resolution : 0.035s
Nodes: 31 (897.5 n/s)
Some Experiments

Using not-equals:

\[
\begin{array}{cccccccc}
8 & 2 & 6 & 3 & 5 & 9 & 7 & 4 & 1 \\
9 & 3 & 5 & 7 & 1 & 4 & 6 & 8 & 2 \\
1 & 7 & 4 & 8 & 6 & 2 & 9 & 5 & 3 \\
6 & 8 & 9 & 2 & 4 & 1 & 5 & 3 & 7 \\
3 & 5 & 7 & 6 & 9 & 8 & 1 & 2 & 4 \\
4 & 1 & 2 & 5 & 7 & 3 & 8 & 6 & 9 \\
7 & 4 & 8 & 9 & 3 & 5 & 2 & 1 & 6 \\
2 & 6 & 3 & 1 & 8 & 7 & 4 & 9 & 5 \\
5 & 9 & 1 & 4 & 2 & 6 & 3 & 7 & 8
\end{array}
\]

Solutions: 1
Initial propagation : 0.018s
Resolution : 0.035s
Nodes: 31 (897.5 n/s)

Using all-different:

\[
\begin{array}{cccccccc}
8 & 2 & 6 & 3 & 5 & 9 & 7 & 4 & 1 \\
9 & 3 & 5 & 7 & 1 & 4 & 6 & 8 & 2 \\
1 & 7 & 4 & 8 & 6 & 2 & 9 & 5 & 3 \\
6 & 8 & 9 & 2 & 4 & 1 & 5 & 3 & 7 \\
3 & 5 & 7 & 6 & 9 & 8 & 1 & 2 & 4 \\
4 & 1 & 2 & 5 & 7 & 3 & 8 & 6 & 9 \\
7 & 4 & 8 & 9 & 3 & 5 & 2 & 1 & 6 \\
2 & 6 & 3 & 1 & 8 & 7 & 4 & 9 & 5 \\
5 & 9 & 1 & 4 & 2 & 6 & 3 & 7 & 8
\end{array}
\]

Solutions: 1
Building time : 0.058s
Resolution : 0.027s
Nodes: 1 (36.4 n/s)
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

Using not-equals:

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

Solutions: 1
Initial propagation : 0.017s
Resolution : 0.031s
Nodes: 14 (457.9 n/s)
Some Experiments

Using not-equals:

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

Solutions: 1
Initial propagation : 0.017s
Resolution : 0.031s
Nodes: 14 (457.9 n/s)

Using all-different:

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

Solutions: 1
Initial propagation : 0.026s
Resolution : 0.034s
Nodes: 2 (59.5 n/s)
Some Experiments

The Telegraph

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
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

Solutions: 1
Initial propagation : 0.016s
Resolution : 0.075s
Nodes: 484 (6,444.8 n/s)
Some Experiments

Using not-equals:

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

Solutions: 1
Initial propagation : 0.016s
Resolution : 0.075s
Nodes: 484 (6,444.8 n/s)

Using all-different:

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

Solutions: 1
Initial propagation : 0.018s
Resolution : 0.073s
Nodes: 180 (2,476.5 n/s)
Some Experiments

Using not-equals:

Don’t know, I gave up after two days.
Some Experiments

Using not-equals:

Don’t know, I gave up after two days.

Using all-different:

Solutions: 1
Initial propagation : 0.305s
Resolution : 0.921s
Nodes: 351 (381.1 n/s)
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.
  - Difficult homework: $a + b = c$ using binary constraints.
- 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.
  - Really easy homework: detecting all-different is NP-hard.
Some Other Interesting Global Constraints

- All different except 0.
- Global cardinality.
- At least, at most, among.
- $n$ Value.
- Regular.
Are You Smarter than a Constraint Solver?

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

\[
\begin{align*}
x_1 & \in \{4, 5\} & x_2 & \in \{1, 2, 3, 4\} & x_3 & \in \{3, 4, 5\} \\
x_4 & \in \{5, 6\} & x_5 & \in \{3, 5\} \\
\text{alldifferent}(x_1, x_2, x_3, x_4, x_5)
\end{align*}
\]

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.