Modeling Sudoku puzzles with Python

Sean Davis  Matthew Henderson  Andrew Smith

June 30, 2010

SciPy 2010
Python for Scientific Computing Conference

June 28 to July 3, 2010
Background - The OKlibrary

- http://www.ok-sat-library.org/
Background - The OKlibrary

- http://www.ok-sat-library.org/
- Open-source research platform and generative library for generalised SAT solving
Background - The OKlibrary

- http://www.ok-sat-library.org/
- Open-source research platform and generative library for generalised SAT solving
- Developed by Oliver Kullmann at the University of Swansea since 2005.
Background - The OKLibrary

- [http://www.ok-sat-library.org/](http://www.ok-sat-library.org/)
- Open-source research platform and generative library for generalised SAT solving
- Developed by Oliver Kullmann at the University of Swansea since 2005.
- Modeling of combinatorial puzzles via SAT (Latin squares/Sudoku)
Background - The OKLibrary

- http://www.ok-sat-library.org/
- Open-source research platform and generative library for generalised SAT solving
- Developed by Oliver Kullmann at the University of Swansea since 2005.
- Modeling of combinatorial puzzles via SAT (Latin squares/Sudoku)
- C++/Lisp/Bash
Background - sudoku.py

▶ http://bitbucket.org/matthew/scipy2010
Background - sudoku.py

- http://bitbucket.org/matthew/scipy2010
- Open-source
Background - `sudoku.py`

- Open-source
- Developed by authors at Berea College since 2010
Background - sudoku.py

- http://bitbucket.org/matthew/scipy2010
- Open-source
- Developed by authors at Berea College since 2010
- Modeling of Sudoku puzzles in a variety of domains
Background - sudoku.py

- http://bitbucket.org/matthew/scipy2010
- Open-source
- Developed by authors at Berea College since 2010
- Modeling of Sudoku puzzles in a variety of domains
- Python
Overview

- Modeling Sudoku in Python
Overview

- Modeling Sudoku in Python
  - Constraint models
Overview

- Modeling Sudoku in Python
  - Constraint models
  - Graph models
Overview

- Modeling Sudoku in Python
  - Constraint models
  - Graph models
  - Integer programming models
Overview

- Modeling Sudoku in Python
  - Constraint models
  - Graph models
  - Integer programming models
  - Polynomial models
Overview

- Modeling Sudoku in Python
  - Constraint models
  - Graph models
  - Integer programming models
  - Polynomial models
- Using sudoku.py
Overview

- Modeling Sudoku in Python
  - Constraint models
  - Graph models
  - Integer programming models
  - Polynomial models
- Using `sudoku.py`
  - Creating puzzles
Overview

▶ Modeling Sudoku in Python
  ▶ Constraint models
  ▶ Graph models
  ▶ Integer programming models
  ▶ Polynomial models
▶ Using sudoku.py
  ▶ Creating puzzles
  ▶ Solving puzzles
A traditional Sudoku puzzle is a partial assignment of 1, . . . , 9 to the cells of a 9 × 9 grid with the latin property on rows, columns and boxes.
A solution of a Sudoku puzzle is a total assignment which extends the original partial assignment and satisfies the same Latin properties.
A (generalized) Sudoku puzzle of boxsize $n$ is a partial assignment of $1, \ldots, n^2$ to the cells of an $n^2 \times n^2$ grid with the latin property on rows, columns and boxes.
A constraint satisfaction problem (CSP) is a collection of constraints.
A constraint satisfaction problem (CSP) is a collection of constraints.
A constraint restricts the values assigned to certain variables.
A constraint satisfaction problem (CSP) is a collection of constraints. A constraint restricts the values assigned to certain variables. A variable $v$ has an associated domain $D(v)$. 
A constraint satisfaction problem (CSP) is a collection of constraints. A constraint restricts the values assigned to certain variables. A variable \( v \) has an associated domain \( D(v) \). A solution of a CSP is an assignment to the variables which satisfies all the constraints.
For a Sudoku puzzle of boxsize $n$ we have variables

$$x_i \quad 1 \leq i \leq n^4$$

The domain $D(x_i) = \{1, \ldots, n^2\}$. 

$x_i = j$ means that cell $i$ is assigned value $j$.
The AllDifferent constraint forces a set of variables to have mutually different values.
Modeling Sudoku – The AllDifferent constraint

The AllDifferent constraint forces a set of variables to have mutually different values.
Modeling Sudoku – The AllDifferent constraint

The AllDifferent constraint forces a set of variables to have mutually different values. For example, if $n = 2$: 

- Row 1: AllDifferent $(x_1, x_2, x_3, x_4)$
- Column 1: AllDifferent $(x_1, x_5, x_9, x_{13})$
- Box 1: AllDifferent $(x_1, x_2, x_5, x_6)$
The *AllDifferent constraint* forces a set of variables to have mutually different values. For example, if $n = 2$:

- Row 1: $\text{AllDifferent}(x_1, x_2, x_3, x_4)$
The AllDifferent constraint forces a set of variables to have mutually different values. For example, if $n = 2$:

- Row 1: $\text{AllDifferent}(x_1, x_2, x_3, x_4)$
- Column 1: $\text{AllDifferent}(x_1, x_5, x_9, x_{13})$
The AllDifferent constraint forces a set of variables to have mutually different values.
For example, if $n = 2$:

- Row 1: AllDifferent($x_1, x_2, x_3, x_4$)
- Column 1: AllDifferent($x_1, x_5, x_9, x_{13}$)
- Box 1: AllDifferent($x_1, x_2, x_5, x_6$)
The *ExactSum* constraint restricts the values of variables to have a given sum. So, if $x_4 = 3$, we can use the constraint $\text{ExactSum}(x_4, 3)$.
http://labix.org/python-constraint
Developed by Gustavo Niemeyer.
http://labix.org/python-constraint
Developed by Gustavo Niemeyer.

```python
>>> from constraint import Problem
>>> from sudoku import cells, symbols
```
from constraint import Problem
from sudoku import cells, symbols

cp = Problem()
cp.addVariables(cells(2), symbols(2))
Modeling Sudoku – The empty board

```python
>>> from sudoku import \\
cells_by_row, cells_by_col, cells_by_box
```
Modeling Sudoku – The empty board

```python
>>> from sudoku import \
    cells_by_row, cells_by_col, cells_by_box

>>> sudoku.cells_by_row(2)
[[1, 2, 3, 4], [5, 6, 7, 8],
 [9, 10, 11, 12], [13, 14, 15, 16]]
```
Modeling Sudoku – The empty board

```python
>>> from sudoku import \
    cells_by_row, cells_by_col, cells_by_box

>>> sudoku.cells_by_row(2)
[[1, 2, 3, 4], [5, 6, 7, 8],
 [9, 10, 11, 12], [13, 14, 15, 16]]

>>> sudoku.cells_by_col(2)
[[1, 5, 9, 13], [2, 6, 10, 14],
 [3, 7, 11, 15], [4, 8, 12, 16]]
```
Modeling Sudoku – The empty board

```python
>>> from sudoku import \\
    cells_by_row, cells_by_col, cells_by_box

>>> sudoku.cells_by_row(2)
[[1, 2, 3, 4], [5, 6, 7, 8],
 [9, 10, 11, 12], [13, 14, 15, 16]]

>>> sudoku.cells_by_col(2)
[[1, 5, 9, 13], [2, 6, 10, 14],
 [3, 7, 11, 15], [4, 8, 12, 16]]

>>> sudoku.cells_by_box(2)
[[1, 2, 5, 6], [3, 4, 7, 8],
 [9, 10, 13, 14], [11, 12, 15, 16]]
```
Modeling Sudoku – The empty board

```python
>>> for row in cells_by_row(2):
    ...   cp.addConstraint(AllDifferentConstraint(), row)
    ...  
```

>>> for row in cells_by_row(2):
    ...     cp.addConstraint(AllDifferentConstraint(), row)
    ...

>>> for col in cells_by_col(2):
    ...     cp.addConstraint(AllDifferentConstraint(), col)
    ...
Modeling Sudoku – The empty board

```python
>>> for row in cells_by_row(2):
    ...    cp.addConstraint(AllDifferentConstraint(), row)
    ...

>>> for col in cells_by_col(2):
    ...    cp.addConstraint(AllDifferentConstraint(), col)
    ...

>>> for box in cells_by_box(2):
    ...    cp.addConstraint(AllDifferentConstraint(), box)
```
>>> d = {3: 2, 5: 2, 6: 1, 7: 4, \
8: 3, 10: 4, 12: 2, 13: 1}
Modeling Sudoku – Puzzles

```python
>>> d = {3: 2, 5: 2, 6: 1, 7: 4, \
     8: 3, 10: 4, 12: 2, 13: 1}
>>> from constraint import ExactSumConstraint as Exact
>>> for cell in d:
...   cp.addConstraint(Exact(d[cell]), cell)
```
Modeling Sudoku – Solving

```python
>>> cp.getSolution()
{1: 4,
  2: 3,
  3: 2,
  4: 1,
  5: 2,
  6: 1,
  7: 4,
  8: 3,
  9: 3,
  10: 4,
  11: 1,
  12: 2,
  13: 1,
  14: 2,
  15: 3,
  16: 4}
```
from sudoku import Puzzle
Modeling Sudoku – Puzzle objects

```python
>>> from sudoku import Puzzle

>>> Puzzle(cp.getSolution(), 2)
```

```
+-----+-----+-----+
| 4 3 | 2 1 |     |
| 2 1 | 4 3 |     |
+-----+-----+-----+
| 3 4 | 1 2 |     |
| 1 2 | 3 4 |     |
+-----+-----+-----+
```
Modeling Sudoku – The solve function

>>> p = Puzzle(d, 2)
Modeling Sudoku – The solve function

>>> p = Puzzle(d, 2)

>>> from sudoku import solve
>>> solve(p)

+-------+-------+-------+
| 4 3   | 2 1   |       |
| 2 1   | 4 3   |       |
+-------+-------+-------+
| 3 4   | 1 2   |       |
| 1 2   | 3 4   |       |
+-------+-------+-------+
Modeling Sudoku – Graph models

Modeling Sudoku – Graph models


- The graph model has a *node* for every cell. Two nodes are adjacent in the graph model if they represent *dependent cells*. 
Modeling Sudoku – Graph models

Figure: The Shidoku graph
Modeling Sudoku – Graph models

Networkx: http://networkx.lanl.gov/

```python
>>> from networkx import Graph
>>> g = Graph()
>>> g.add_nodes_from(cells(2))
```
Modeling Sudoku – Graph models

Networkx: http://networkx.lanl.gov/

```python
>>> from networkx import Graph
>>> g = Graph()
>>> g.add_nodes_from(cells(2))
```

```python
>>> from sudoku import dependent_cells
>>> g.add_edges_from(dependent_cells(2))
```
Modeling Sudoku – Node coloring

```python
>>> for cell in d:
    ...     g.node[cell][‘color’] = d[cell]
```
Modeling Sudoku – Node coloring

```python
>>> for cell in d:
    g.node[cell]['color'] = d[cell]

>>> from sudoku import node_coloring, n_colors
>>> cg = node_coloring(g)
>>> n_colors(cg)
6
```
Modeling Sudoku – Node coloring

```python
>>> for cell in d:
    ...     g.node[cell][’color’] = d[cell]

>>> from sudoku import node_coloring, n_colors
>>> cg = node_coloring(g)
>>> n_colors(cg)
6

>>> from sudoku import graph_to_dict
>>> s = Puzzle(graph_to_dict(cg), 2)
>>> s
+-----+-----+
| 3 5 | 2 6 |
| 2 1 | 4 3 |
+-----+-----+
| 5 4 | 3 2 |
| 1 2 | 5 4 |
+-----+-----+
```
### Modeling Sudoku – Further models

<table>
<thead>
<tr>
<th>Modeling domain</th>
<th>Python library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Programming</td>
<td>pyglpk v0.3</td>
</tr>
<tr>
<td></td>
<td><a href="http://tfinley.net/software/pyglpk/">http://tfinley.net/software/pyglpk/</a></td>
</tr>
<tr>
<td>Polynomials</td>
<td>sympy</td>
</tr>
</tbody>
</table>