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/`
- 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`

- `http://bitbucket.org/matthew/scipy2010`
- 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 $1, \ldots, 9$ to the cells of a 9×9 grid with the latin property on rows, columns and boxes.

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

A *solution* of a Sudoku puzzle is a total assignment which extends the original partial assignment and satisfies the same latin properties.

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

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.

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

Constraint Satisfaction Problems

A *constraint satisfaction problem* (CSP) is a collection of *constraints*.

Constraint Satisfaction Problems

A *constraint satisfaction problem* (CSP) is a collection of *constraints*.
A *constraint* restricts the values assigned to certain *variables*.

Constraint Satisfaction Problems

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* $D(v)$.

Constraint Satisfaction Problems

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.

Modeling Sudoku – Variables

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.

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.

Modeling Sudoku – The AllDifferent constraint

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

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)

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})

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)

Modeling Sudoku – The ExactSum constraint

The *ExactSum* constraint restricts the values of variables to have a given sum.

So, if $x_4 = 3$, we can use the constraint ExactSum$(x_4, 3)$

Modeling Sudoku – python-constraint

`http://labix.org/python-constraint`

Developed by Gustavo Niemeyer.

Modeling Sudoku – python-constraint

http://labix.org/python-constraint

Developed by Gustavo Niemeyer.

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

Modeling Sudoku – python-constraint

http://labix.org/python-constraint

Developed by Gustavo Niemeyer.

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

>>> cp = Problem()
>>> cp.addVariables(cells(2), symbols(2))
```

Modeling Sudoku – The empty board

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

Modeling Sudoku – The empty board

```
>>> 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

```
>>> 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

```
>>> 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

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

Modeling Sudoku – The empty board

```
>>> 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

```
>>> 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)
```

Modeling Sudoku – Puzzles

```
>>> d = {3: 2, 5: 2, 6: 1, 7: 4, \
         8: 3, 10: 4, 12: 2, 13: 1}
```

Modeling Sudoku – Puzzles

```
>>> 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

```
>>> 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}
```

Modeling Sudoku – `Puzzle` objects

```
>>> from sudoku import Puzzle
```

Modeling Sudoku – `Puzzle` objects

```
>>> 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

- J. Gago-Vargas, I. Hartillo-Hermosa, J. Martin-Morales, J. M. Ucha- Enriquez, *Sudokus and Groebner Bases: not only a Divertimento*, In: Lecture Notes in Computer Science, vol. 4194. pp. 155-165. 2005

Modeling Sudoku – Graph models

- J. Gago-Vargas, I. Hartillo-Hermosa, J. Martin-Morales, J. M. Ucha- Enriquez, *Sudokus and Groebner Bases: not only a Divertimento*, In: Lecture Notes in Computer Science, vol. 4194. pp. 155-165. 2005
- 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

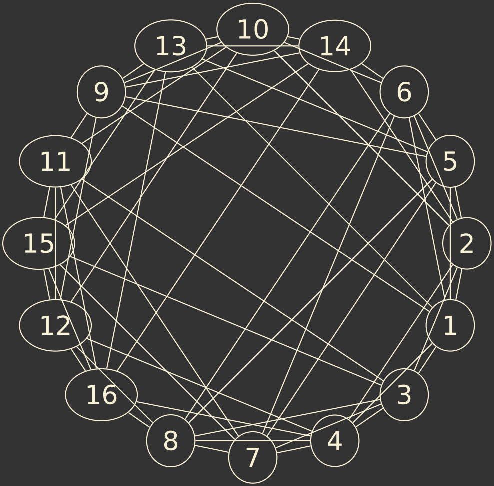

Figure: The Shidoku graph

Modeling Sudoku – Graph models

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

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

Modeling Sudoku – Graph models

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

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

>>> from sudoku import dependent_cells
>>> g.add_edges_from(dependent_cells(2))
```

Modeling Sudoku – Node coloring

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

Modeling Sudoku – Node coloring

```
>>> 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

```
>>> 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

Modeling domain	Python library
Integer Programming	`pyglpk v0.3`
	`http://tfinley.net/software/pyglpk/`
Polynomials	`sympy`
	`http://code.google.com/p/sympy/`