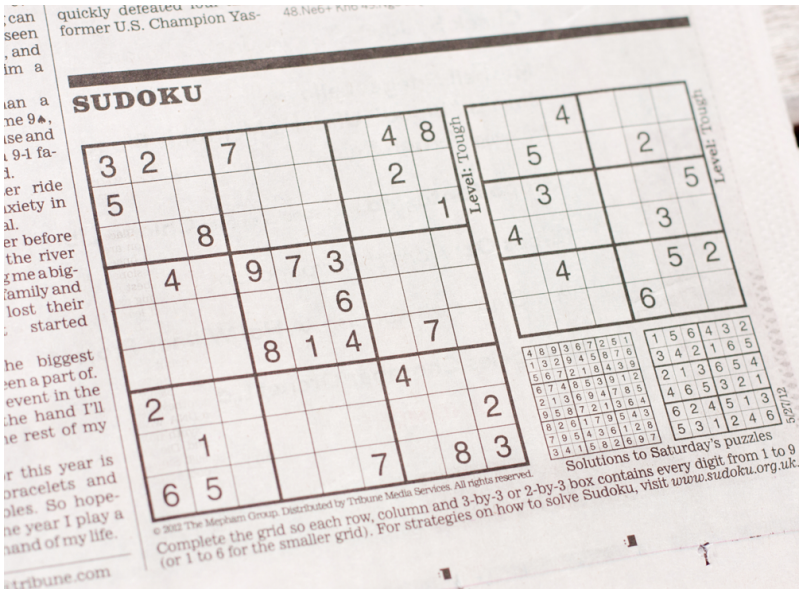
Solving Single-digit Sudoku Subproblems

David Eppstein

Int. Conf. Fun with Algorithms, June 2012

Sudoku

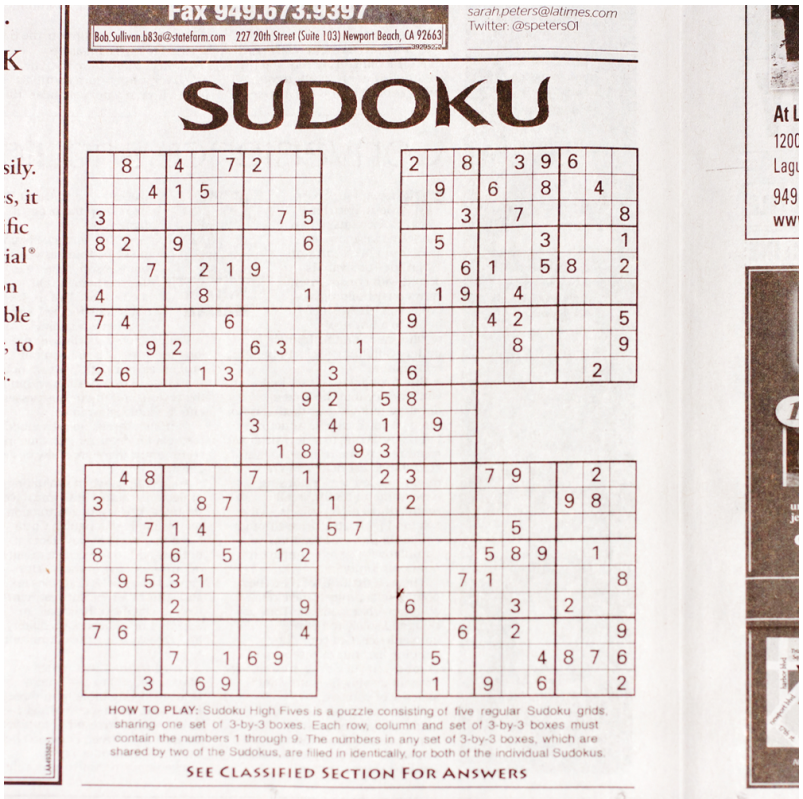


Newspaper images from *L.A. Times*, May 27, 2012

An $ab \times ab$ array of cells, partitioned into $a \times b$ blocks, partially filled in with numbers from 1 to ab.

Must fill in remaining cells so that each number appears exactly once in each row, column, and block

Sudoku variations



Commonly used sizes for Sudoku puzzles include 6×6, 9×9, and 16×16

Many other variations such as *Samurai Sudoku* (left), formed by five overlapping 9×9 Sudoku puzzles that must be solved simultaneously.

A brief history of Sudoku

Similar puzzles of finishing partial magic squares
go back to the 19th century

Modern Sudoku was invented by Howard Garns
and published in 1979 as "Number Place"

Introduced to Japan in 1984 as "Suji wa dokushin ni kagiru"
("the digits must be single")
later abbreviated as Sudoku

Brought back from Japan to U.S. and Europe in 2004–2005

Now commonly found in newspapers, on the web,
in smartphone apps, etc.

From Wikipedia, http://en.wikipedia.org/wiki/Sudoku

Human vs computer problem solving

Humans

- Solve the puzzle one step at a time without backtracking
- Each step involves either logical deduction or (more often) matching known patterns
- Solution must be unique; some deduction patterns make use of that knowledge

Computers

- Can solve most puzzles very quickly by simple backtracking techniques
- Nevertheless, Sudoku is NP-hard in general [Yato & Seta 2003]

  (The assumption of a unique solution complicates its complexity class.)

Making computers work more like humans

Instead of backtracking, implement a repertoire of deductive rules

Repeatedly search for a rule that fits the puzzle and apply it until either the puzzle is solved or the solver is stuck.

Slower and less effective than backtracking, so why?

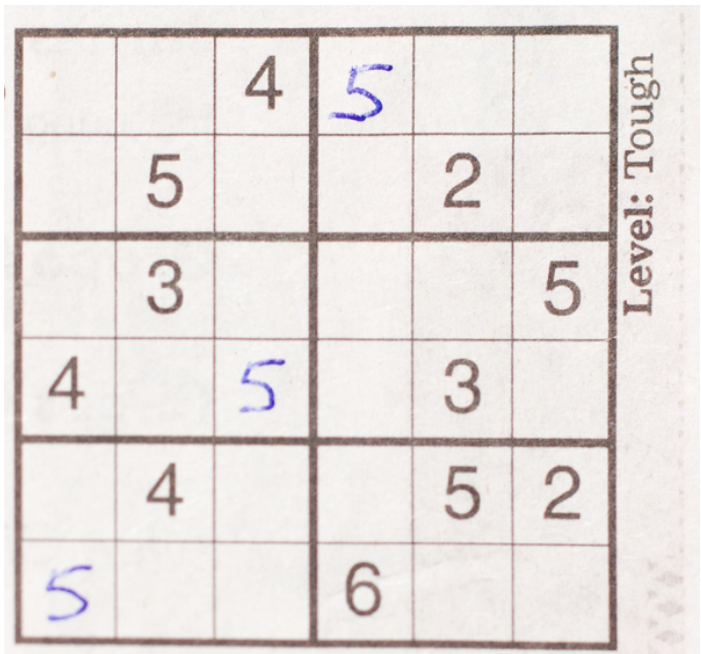
- Automatically grade puzzle difficulty
  (more complex deductions mean a more difficult puzzle)
- Provide insight into human problem solving abilities
- Explain solution to a human learner

An example

In this "tough" 6×6 example, the first few deductions are easy:

The top and middle 5's are the only possible location for a 5 in their rows

The bottom 5 is the only possible location for a 5 in its column



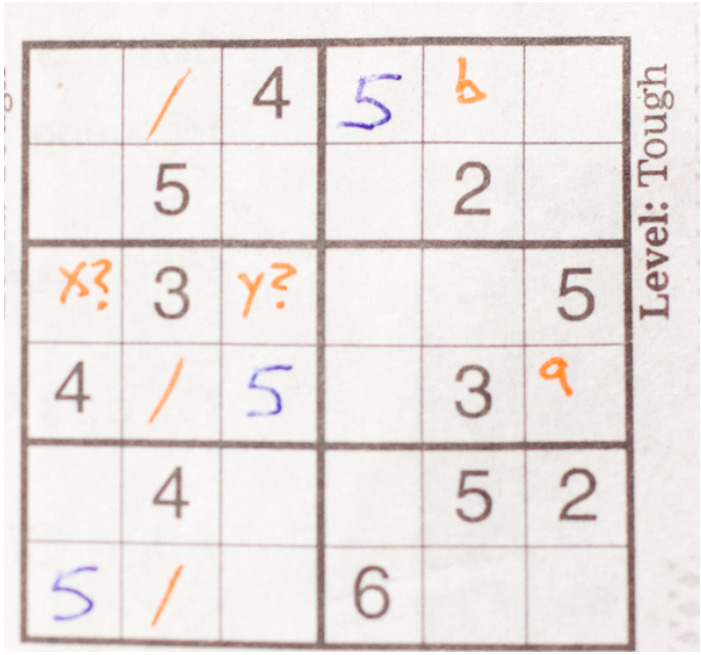
An example

Where can the 6's go?

Suppose that we place a 6 in either cell x or cell y

Then a becomes the only possible location for a 6 in its row

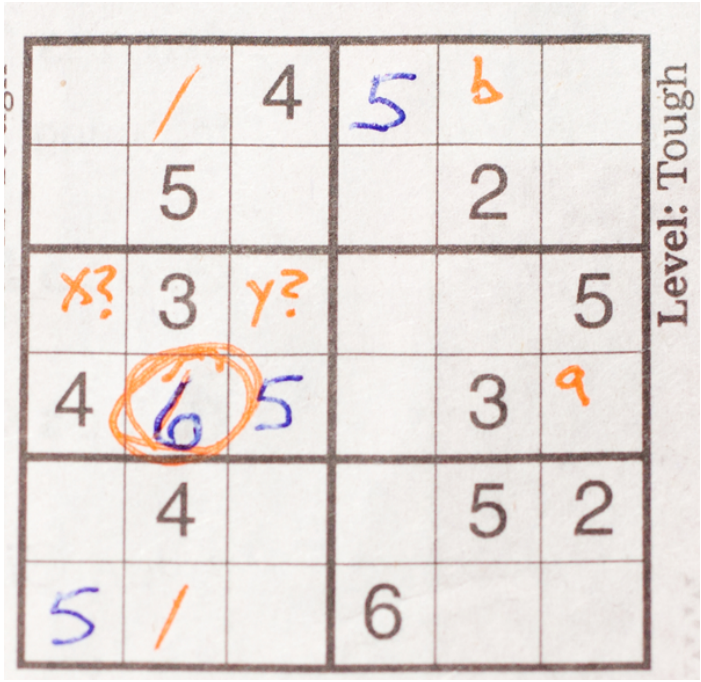
And b becomes the only possible location for a 6 in its column



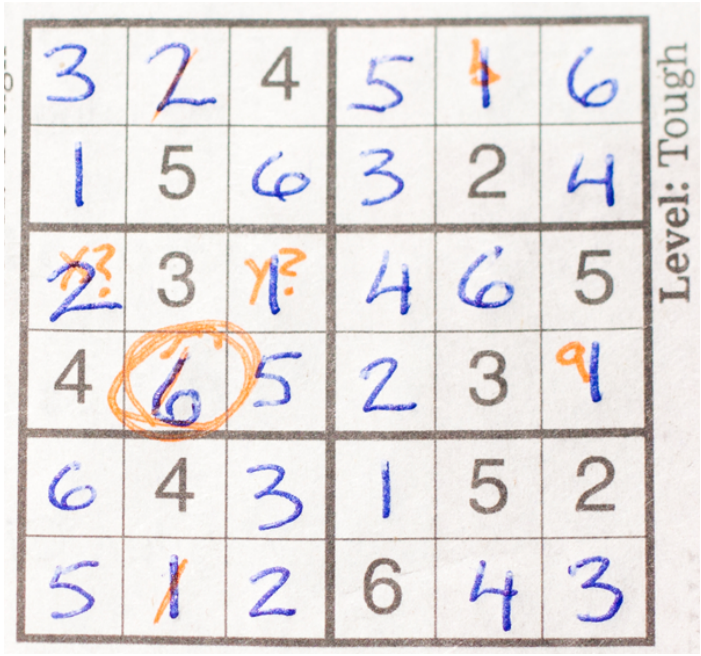
But after these choices, there is nowhere available to put a 6 in the second column

So neither x nor y is possible

An example



There is only one remaining location for a 6 in the left middle block



Once that digit is placed, the remaining deductions are easy

Nishio

Steps in which we only look at one digit at a time
(in the example, first 5's, then 6's)
and make all possible deductions involving only that digit
are called *Nishio* (after Tetsuya Nishio).

We are given a set of potential placements for a digit (as determined by previous placements and deductions)

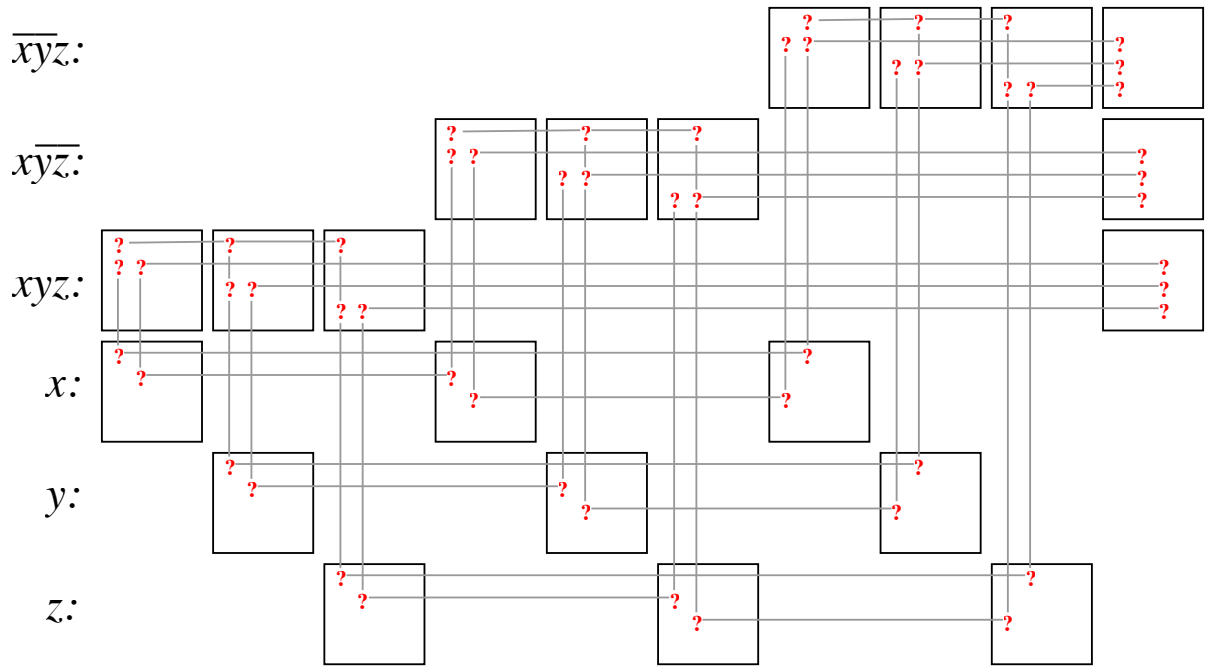
Which cells in the set can be part of a *valid placement* that includes one cell in each row, column, and block?

Which cells must be part of all valid placements?

A complex deduction rule but very powerful

How hard is Nishio deduction?

NP-complete, by reduction from 3-SAT



So the best we can hope for is an exponential time algorithm

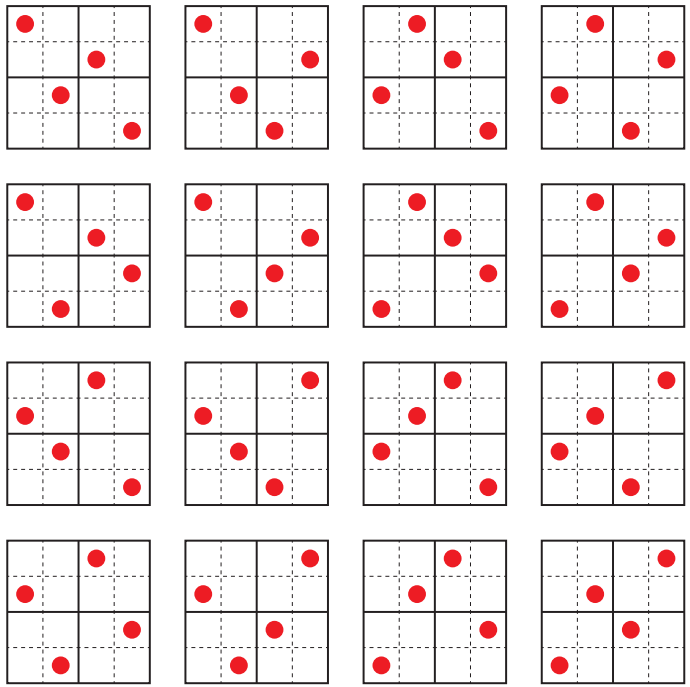
But some exponential algorithms are more practical than others...

Best previously-used algorithm

Pattern overlay method:

- Precompute list of valid placements
- Test whether each uses cells still available to the given digit
- Compute union of the available placements

$ab \times ab$ Sudoku
has $a!^b b!^a$ placements



All 16 valid placements
for 4×4 Sudoku

688 for 6×6, 46656 for 9×9, 110075314176 for 16×16, ...

Main idea of new algorithm

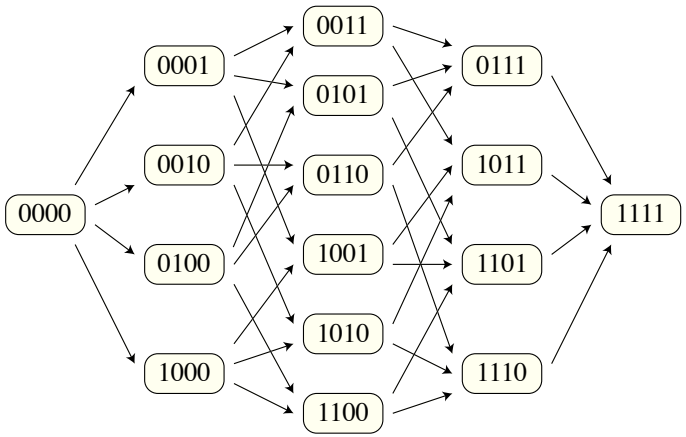
Precompute a DAG in which

- Edges correspond to puzzle cells
- Paths from source s to sink t correspond to valid placements

Form subgraph of edges
that come from cells available to the given digit

Use DFS-based reachability analysis
to find edges that belong to s–t paths

A graph that almost works



n-dimensional hypercube (where puzzle is $n \times n$)

2^n vertices (n-bit numbers)

Edge = two numbers that differ in a single bit

Puzzle cell in row i, column j corresponds to edges at distance j from $\vec{0}$ that change the ith bit from 0 to 1

Every path from $\vec{0}$ to $\vec{1}$ gives a placement with

- One cell per row (one edge that sets bit i from 0 to 1)
- One cell per column (one edge at distance j from $\vec{0}$)

But what about the constraint of having only one cell per block?

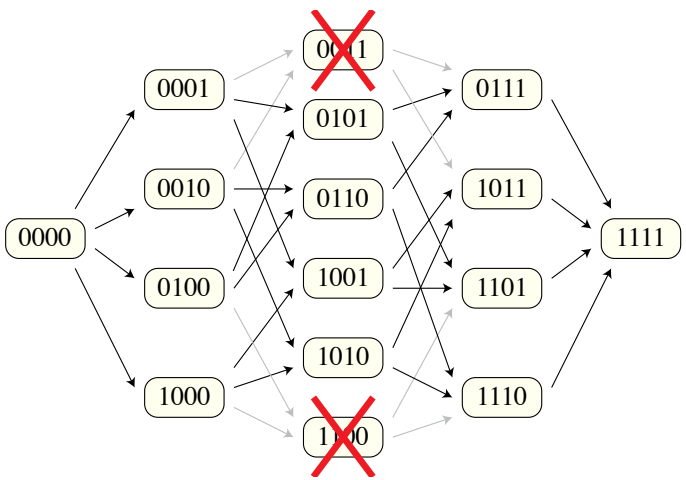
Eliminating the bad paths

Instead of n-bit binary numbers, use $b \times a$ binary matrices

Puzzle rows in the same block $\Leftrightarrow$ bits in the same matrix row

Vertex can be part of a valid placement $\Leftrightarrow$ matrix is *balanced*
(numbers of nonzero bits in all rows are within ± 1 of each other)

Delete unbalanced matrix vertices from hypercube



Vertex 0011 $\Rightarrow \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$
gives placements with two cells in bottom left block and two in top right block

Similarly 1100 gives two cells in top left block etc

Paths in remaining graph correspond to valid placements as desired

Analysis of the new algorithm

Total time is within a polynomial factor of the number of graph vertices = the number of $b \times a$ balanced matrices

So how many can there be?

290 for 9×9 Sudoku, 19442 for 16×16 Sudoku

In general,

$$\sum_{i=0}^{b-1} \left(\binom{b}{i} + \binom{b}{i+1} \right)^a - \sum_{i=1}^{b-1} \binom{b}{i}^a$$

(i = smaller number of nonzeros in balanced matrix rows; second sum corrects double counting when all rows have equal nonzeros)

Stirling's formula $\Rightarrow 2^{n-\Omega(\sqrt{n}\log n)}$

Conclusions

New algorithm for important subproblem in human-like Sudoku

Scales singly exponentially instead of factorially

Simple, implemented, works well in practice

Even for 9×9 should be much faster than pattern overlay

Open: can we solve full Sudoku puzzles in $2^{o(n^2)}$?

More generally, many more problems to be studied in exponential-time algorithms for puzzles and games