Sudoku and Other Related Problems

E. Routledge

01 Nov 2022

1 Introduction

Sudoku is a simple logic game, in the standard 9×9 (or $3 \times 3 \times 3 \times 3$) one must complete the grid such that every row, column and box contains the numbers 1 to 9, that is all. Yet it leads to interesting and difficult puzzles.

1.1 History

2 Sudoku is Hard

Sudoku can be described in a single rule but it is in fact a hard problem to solve. Instances of the puzzle requiring complex x wing and y wing strategies are not what makes the puzzle hard to solve, it is hardness through a provable, computational lense for which this section cares.

2.1 Computational Complexity an Introduction

What is NP completeness? We care about decision problems, these are problems that given an input produce a 'yes' or 'no' answer. We care about three of the subsets of these problems: P is the the class of problems that can be solved in polynomial time by a Turing machine; NP is the class of problems that can be verified in polynomial time and solved in polynomial time by a non-deterministic Turing machine, finally, the NP-complete set has problems that any NP problem can be reduced to in polynomial time (if there is a NP-complete problem that can be solved in polynomial time then P=NP, this is one of the millenium prize problems). This is to say NP problems are hard, it is assumed they cannot be solved in polynomial time ($P \neq NP$) and are therefore infeesible for large inputs.

How to prove NP completeness generally? Show verifier has a polynomial or less runtime. Reduce a known NP-complete problem to the

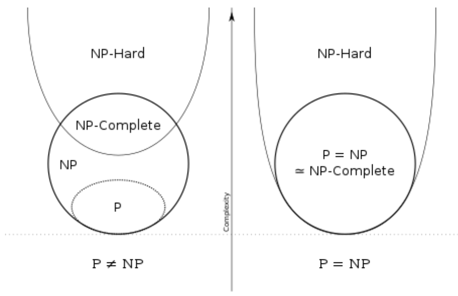

Figure 1: P, NP, NP-complete & NP-hard sets [1]

problem you wish to prove is NP-complete (call this x), one does this by transforming the input of a known NP-complete problem to the input x. This proves that the x is at least as hard as the NP-complete problem as if we had access to an polynomial time algorithm to solve x we could solve the known NP-complete problem in polynomial time too creating a contradiction.

Our base NP-complete problem. If, as the above suggests, we require a NP-complete problem to prove a problem is NP-complete then we seem to have reached a paradox. Luckily we have the Cook-Levin Theorem.

Cook-Levin Theorem: SAT is NP-Complete

Proof: cite

Definition: SAT is the following decision problem. Given a set of boolean variables B and a collection of clauses C does a valid truth assignment exist that satisfies C?

We now have a NP-complete problem to reduce other problems to.

2.2 Validation is Easy

Valid Grid decision problem:

$$\Psi(\text{grid}(n^2, n^2)) = \begin{cases} \text{True if the grid is valid} \\ \text{False if the grid is invalid.} \end{cases} \tag{1}$$

There exists an algorithm to do this in polynomial time with respect to the dimensions of the grid.

1. For each row in the grid check there exists no repeated numbers. $O(n^2)$
2. For each column in the grid check there exists no repeated numbers. $O(n^2)$
3. For each box in the grid check there exists no repeated numbers. $O(n^2)$

If all tests pass return True else return False. This algorithm has complexity of $O(n^2 + n^2 + n^2) = O(3n^2) = O(n^2)$, this is polynomial and therefore $\Psi(\text{grid}) \in P$.

2.3 Finding a Solution is Hard

Finding a solution to sudoku is NP-complete, let us define the decision problem:

$$\Phi(\text{grid}(n^2, n^2)) = \begin{cases} \text{True if a solution exists} \\ \text{False if a solution does not exist.} \end{cases} \tag{2}$$

Our question now is does there exist a function Φ that when given an instance of the problem (in this case a grid of numbers and blanks) will, in polynomial time or less return True if it can be solved and False otherwise.

Proof

The **verifier** is $O(n^2)$, as will be seen in the above subsection 'Validation is Easy', so the problem is in the NP set.

Now we need a **reduction** from sudoku to a known NP-complete problem to prove sudoku is at least as hard. We will be creating a chain of reductions: Sudoku $\geq_p$ Latin Square $\geq_p$ Triangulated Tripartite $\geq_p$ 3SAT and then prove 3SAT is NP-complete.

Sudoku $\geq_p$ Latin Square

To reduce a given sudoku grid of size $n^2 \times n^2$ to a latin square (definition of which see in Other Related Problems - Latin Squares) take the first row of boxes and select the first column from each box, these will make a $n \times n$ latin square. If any numbers are repeated we can relabel without a problem

Latin Square $\geq_p$ Triangulated Tripartite

Triangulated Tripartite $\geq_p$ 3SAT

3SAT is NP-Complete

What is 3SAT? With a set of boolean variables B and a collection of clauses C, with at most 3 literals (a literal is any $b \in B$ or its negation $\bar{b}$) in each, does a valid truth assignment exist that satisfies C?

$$\phi(C, B) = \begin{cases} \text{True if a truth assignment exists} \\ \text{False if a truth assignment does not exist.} \end{cases} \tag{3}$$

This decision problem is therefore an enforced limitation of SAT as defined in the subsection Computational Complexity an Introduction.

Proof Given a truth assignment t check each clause is satisfied, if all are satisfied return True else False, this algorithm is at most the length of C

multiplied by the length of B. $O(BC)$ is polynomial, a polynomial verifier exists.

Given a SAT instance with the input sets of B and C. C is in conjunctive normal form (every clause set can be converted to an equivalent set in CNF form [2]) such that $\forall c \in C$ and for some $b_1, ..., b_n \in B$, $c = b_1 \vee b_2 \vee ... \vee b_n$. For each $c \in C$ with more than 3 literals we can transform these to a new set of clauses of length 3.

For $c = b_1 \vee b_2 \vee ... \vee b_n$ we introduce a new literal: a_1 to give $b_1 \vee b_2 \vee a_1$, $\bar{b_1} \vee a_1$, $\bar{b_2} \vee a_1$ and $a_1 \vee b_3 \vee ... \vee b_n$. Then $a_1 \vee b_3 \vee ... \vee b_n$ becomes $b_3 \vee b_4 \vee a_2$, $\bar{b_3} \vee a_2$, $\bar{b_4} \vee a_2$ and $a_1 \vee a_2 \vee b_5 \vee ... \vee b_n$. This continues at most $n/2$ times to give $a_1 \vee ... \vee a_{n/2}$ or $a_1 \vee ... \vee a_{n/2} \vee b_n$ if n is odd.

Because we can convert a clause larger than 3 into multiple clauses of at most 3 literals in linear time ($O(n/2 + n/4 + ...) = O(n)$) this means we can reduce SAT to 3SAT in polynomial time.

As SAT is NP-complete by the Cook-Levin Theorem, this proves 3SAT is NP-Complete. $\square$

Alternative reduction Sudoku $\geq_p n^2$ Graph Colouring.

2.4 Determining Uniqueness is Hard

It is hard to determine if a puzzle has a unique solution?

3 Other Related Problems

3.1 Latin Squares

- A latin square is an n by n matrix filled with n characters that must not repeat along columns or rows.
- Reduced Form - f first row and column is in the natural order
- Equivalence classes
- Number of n by n latin squares is bounded
- Latin squares can be considered a bipartite graph
- Agronomic Research
- Latin hypercube

3.2 Magic Squares

- A magic square is a matrix of numbers with each column, row and diagonal summing to the same value, this value is known as a magic constant and the degree is the number of columns/rows.
- A normal magic square is one containing the integers 1 to n^2.
- Magic Squares with repeating digits are considered trivial.
- Semimagic squares omit the diagnonal sums also summing to the magic constant.
- Truly thought to be magic Shams Al-ma'arif.
- Generation, there exists not completely general techniques. Diamond Method
- Associative Magic Squares
- Pandiagonal Magic Squares
- Most-Perfect Magic Squares
- Equivalence classes for $n <= 5$ but not for higher orders.
- The enumeration of most perfect magic squares of any order.
- 880 distinct magic squares of order four
- Normal magic squares can be constructed for all values except 2
- Preserving the magic property when transformed
- Methods of construction
- Multiplicative magic squares - produce infinite
- Sator square
- magic square of squares - Parker Square is a failed example of this

3.3 Greco-Latin Squares

- Two orthogonal latin squares super imposed, such that the pairs of values are unique.
- Group based greco latin squares
- Eulers interest came from construction of magic squares
- Exists for all but 2 and 6.

4 Solving Techniques

4.1 Backtracking

The standard way to solve a 9×9 sudoku puzzle is by the backtracking algorithm. This is a brute force method with a few optimisations. One can expect to find this algorithm in a computer science course introduction to recursion, that is to say it is not a complex concept and while useful for the usual sizes, as soon as we increase to 16×16 this becomes infeesible. Multiplication tables of quasigroups. Orthogonal latin squares are used in error correcting codes.

Listing 1: Backtracking

```
def Backtracking(grid):
    for each row:
        for each column:
            if grid is empty at this potion:
                try a value in this position
                Backtracking(grid with new value)
                if successful:
                    return grid
                else:
                    try another value
                if no values left to try:
                    return False
    return grid
```

Why does brute force not work for larger examples?

4.2 Stochastic Methods

4.2.1 Simulated Annealing

4.2.2 Genetic Algorithm

5 Generating Techniques

A polynomial generation algorithm without requiring a uniqueness checker which we have proven to be np-complete and therefore infeesible for large n.

6 17 is the Magic Number

4 for shidoku

6.1 Sparsity - information theory

Bomb sudoku/latin squares - Additional rule: the same number can not occur in adjacent or diagonally adjacent squares.

7 Group theory

7.1 Starting Simple

Let us use Shidoku which is specifically a sudoku with $n = 2$.

Only 2 fundamentally different. One has 96 identical, other has 192. Why not the same amount?

7.2 Equivalence Classes

8 Topology

8.1 Torus

9 Constraint Programming

Use of polynomials Roots of unity Grobner Basis

References

[1]

[2] Artificial Intelligence: A modern Approach Archived 2017-08-31 at the Wayback Machine [1995...] Russell and Norvig

[3]

[4]

[5] https://onlinelibrary.wiley.com/doi/10.1002/(SICI)1520-6610(1996)4:6¡405::AID-JCD3¿3.0.CO;2-J

[6] http://joas.agrif.bg.ac.rs/archive/article/59

[7] https://www.semanticscholar.org/paper/Permutation-arrays-for-powerline-communication-and-Colbourn-Kløve/7e69cfdbd2082463c66de698da1e326f0556d1d4

[8] http://www.multimagie.com/English/SquaresOfSquaresSearch.htm

[9] https://plus.maths.org/content/anything-square-magic-squares-sudoku

[10] https://link.springer.com/book/10.1007/978-1-4302-0138-0