Sudoku is Hard

Eve Routledge

Durham University, Supervisor: Prof. Peter Craig

Github:

Sudoku: An Introduction

Sudoku is the classic puzzle where one is given a 9 grid and must fill in the blank entries with numbers 1 to 9, such that there is no repeated number in the row, column or sub square.

Let's imagine a sudoku of size $D^2 \times D^2$. How big does D have to be for you to need more than a day to solve it? Maybe 6 or 10 or even just 4. Don't worry if you said a smaller number than your friends, this has nothing to do with your problem solving skills, even a computer finds sudoku hard. In fact just incrementing D by 1 leads to an exponential increase in compute time and the most optimal algorithms for solving sudoku are infeasible for 100×100.

We prove sudoku's hardness by transforming it into a known 'difficult' problem; we will use SAT, a problem that has plagued computer scientists for decades.

We do this in 4 steps:

Sudoku $\geq_p$ Latin Square $\geq_p$ Triangulate a Tripartite Graph $\geq_p$ 3-SAT $\geq_p$ SAT.

Def^n: A valid **Sudoku** puzzle is a function $S : i, j \to x$ for values $i, j \in \{1, ..., D^2\}$ and $x \in \{0, ..., D^2\}$ satisfying the following:

- for all $a, b, c \in \{1, ..., D^2\}$ with $S(a, b) \neq 0$ and $S(a, c) \neq 0$, then $S(a, b) \neq$ S(a,c)
- for all $a, b, c \in \{1, ..., D^2\}$ with $S(a, b) \neq 0$ and$S(c, b) \neq 0$, then $S(a, b) \neq$ S(c,b)
- for all $a, b, c, d \in \{1, ..., D^2\}$ with a mod $D = c$ mod D, b mod $D = d$ mod D, $S(a, b) \neq 0$ and $S(c, d) \neq 0$, then $S(a, b) \neq S(a, c)$

It is completed if $x \in \{1, ..., D^2\}$.

Def^n: A valid **Latin Square** puzzle is a function $L : i, j \to x$ for values $i, j \in \{1, .., D\}$ and $x \in \{0, ..., D\}$ satisfying the following:

- for all $a, b, c \in \{1, ..., D\}$ with $L(a, b) \neq 0$ and $L(a, c) \neq 0$ then $L(a, b) \neq L(a, c)$
- for all $a, b, c \in \{1, ..., D\}$ with $L(a, b) \neq 0$ and $L(c, b) \neq 0$ then $L(a, b) \neq L(c, b)$

It is complete or solved if for all $i, j \in \{1, ..., D\}$, $L(i, j) \neq 0$.

Def^n: A graph $G = (V, E)$ is **Tripartite** if a partition V_1, V_2, V_3 exists such that the vertices are split into three sets with no edges between vertices that belong to the same set, i.e for all $(v_i, v_j) \in E$ if $v_i \in V_i$ then $v_j \notin V_i$. A **Triangulation** T of a graph G is a way to divide all edges into disjoint subsets T_i, each forming a triangle ($T_i = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$).

Def^n: A **boolean expression** is a formula made of variables $\in B$ and operations (conjunction $\wedge$, disjunction $\vee$, not $\neg$). A literal is $b \in B$ or $\neg b$. A clause $c \in C$ is a disjunction of literals, $b_1 \vee b_2 \vee ...b_i$. A boolean expression is satisfiable if a truth assignment exists such that the expression evaluates to true.

Def^n: SAT is the decision problem determining whether a boolean expression is satisfiable or not. **3-SAT** is an enforced limitation of SAT such that each clause is made up of 3 literals.

Computational Complexity [1]

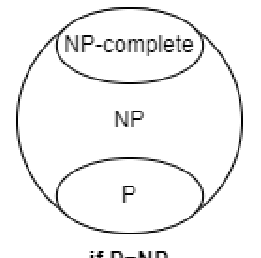
For those with a mathematical mind, outraged by the lack of definitions of 'difficulty' and 'hardness', let's take a detour into complexity theory.

Def^n: Let f be a function indicating the execution time for an algorithm and g a strictly positive function. $f(x) = O(g(x))$ if $\exists$ positive M and x_0 such that $|f(x)| \leq Mg(x) \; \forall \; x \geq x_0$.

Def^n: A **Reduction**, $A \leq_p B$, is a transformation in polynomial time ($O(x^c)$) from problem A to B.

Def^n: A **Turing Machine** is the mathematical model of a CPU.

- P is the set of problems solved in $O(x^c)$ by a Turing machine;
- NP is the set verified in $O(x^c)$ and but not solved in $O(x^c)$;
- the NP-complete set has problems that any NP problem can be reduced to in $O(x^c)$.



if P=NP

Problems in P are considered feasible and those in NP are infeasible as their complexity scales exponentially with respect to the input size. This is if we assume $P \neq NP$ which is of course yet to be proven and remains one of the unsolved Millennium Prize problems.

So when we state sudoku is hard we are actually saying sudoku belongs to NP.

By outlining a reduction SAT $\leq_p$ Sudoku we show sudoku is NP-complete and therefore in NP.

Intuition: Assume we have an algorithm Φ solving sudoku in $O(x^{c_1})$ then if there exists a reduction algorithm Γ of SAT $\leq_p$ Sudoku in $O(x^{c_2})$ then for a given SAT problem x, $\Phi(\Gamma(x))$ will give an algorithm solving SAT in $O(x^{c_1+c_2})$ which, given the Cook-Levin theorem, is a contradiction.

Cook Levin Theorem

SAT is hard.

SAT $\leq_p$ 3 SAT

To begin we look at the bread and butter of complexity theory.

Given a SAT instance with the input sets of B and C. 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.

$(a \vee b) \wedge (\neg a \vee \neg b)$

a	b		$(a \vee b)$	$(\neg a \vee \neg b)$	$(a \vee b) \wedge (\neg a \vee \neg b)$
F	F		F	T	F
F	T		T	T	T
T	F		T	T	T
T	T		T	F	F

Figure 1. Truth Assignment Example with Highlighted Valid Assignment

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 3 SAT in polynomial time.

3SAT is hard. $\square$

3 SAT $\leq_p$ Triangulated Tripartite Graph [3]

This reduction is a little trickier as we need to introduce the Holyer graph H which dips a toe into topology as it is a torus.

Consider the graph $H_{3,p}$, a version of which can be seen in figure 2. Observe $H_{3,p}$ is a tripartite graph iff $p \equiv 0 \; (\text{mod } 3)$, see the 3-colouring ($p$ refers to the 'height' and 'width').

$H_{3,p}$ has only two triangulations, termed a true and a false triangulation, we say $G = T$ if graph G has a true triangulation and $G = F$ if it has a false triangulation. We connect graphs together by taking a set of vertices in G_1 and making them the 'same' as a set of vertices in G_2, sets are the same size.

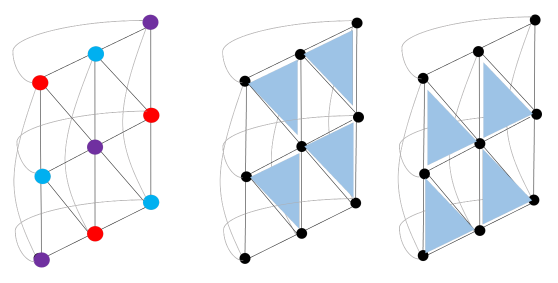


Figure 2. $H_{3,3}$, false, true triangulation

Lemma: Connecting two $H_{3,p}$ by two A-patches then our triangulations of these graphs can be of the form (T, T), (T, F) or (F, T). Then by removing the centre triangles from both graphs we get only the triangulations (T, F) or (F, T). If we expand this to x $H_{3,p}$ we get only one false triangulation and the rest true.

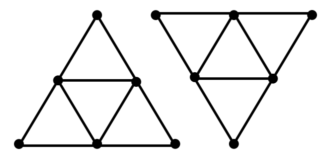


Figure 3. A,B-patch

Transformation process: (select p large enough to prevent patch overlap)

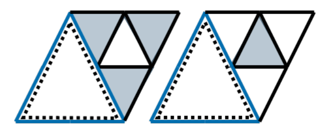


Figure 4. Lemma Illustration

- For $b_i \in B$ create $H_{3,p}$ called G_{b_i}.
- For all $c_j \in C$, for each literal $l_{i,j}$ $i \in [1, 2, 3]$ create $H_{3,p}$ called $G_{i,j}$.
- For $b_i \in B$ connect an A-patch to each $G_{i,j}$ when $l_{i,j} = b_i$ or a B-patch to $G_{i,j}$ when $l_{i,j} = \neg b_i$.
- For each $c_i \in C$ connect an A-patch from $G_{1,i}$, $G_{2,i}$ and $G_{3,i}$ then delete the centre triangle.
- $G = \{G_{b_i} \mid b_i \in B\} \cup \{G_{i,j} \mid c_j \in C \text{ and } i \in [1, .., 3]\}$

Now we can take a 3SAT formula and convert it to the problem of triangulating a tripartite graph.

See figure 5 for an example. By the lemma; one of $l_{1,1}$, $l_{1,2}$ and $l_{1,3}$ must be a false triangulation and therefore the variable connected to this false triangulation must be true, this satisfies the formula.

Triangulating a Tripartite Graph is hard. $\square$

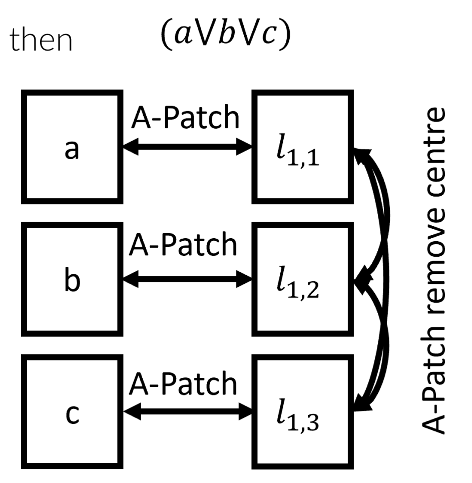


Figure 5. Example

Triangulated Tripartite Graph $\leq_p$ Latin Square [2]

We map a Latin square to a graph through the following: given tripartite graph $G = (V, E)$ label vertices in V_1 with distinct labels $\{r_1, ...r_n\}$, label vertices in V_2 with distinct labels $\{c_1, ...c_n\}$ and label vertices in V_3 with distinct labels $\{e_1, ...e_n\}$. Add edges such that:

- If $L(i, j) = 0$ then add the edge (r_i, c_j)
- If $\forall$ and constant j, $L(i, j) \neq k$ then add the edge (r_i, e_k)
- If $\forall$ and constant i, $L(i, j) \neq k$ then add the edge (c_j, e_k)

1		
2	3	

L

This graph has a triangulation if and only if $L(i, j)$ can be solved.

Let us show every uniform tripartite graph is the above formulation o square.

Def^n: A Latin framework LF for tripartite graph $G = (V, E)$, size (r, s, t) is a $r \times s$ array with values $[1, ..., t]$. With constraints:

- Each row/column contain each element only once.
- If $(r_i, c_j) \in E$ then $LF(i, j) = 0$ else $LF(i, j) = k$ where $k \in [1, ..., t]$
- If $(r_i, e_k) \in E$ then for constant i, $LF(i, j) \neq k$
- If $(c_i, e_k) \in E$ then for constant j, $LF(i, j) \neq k$

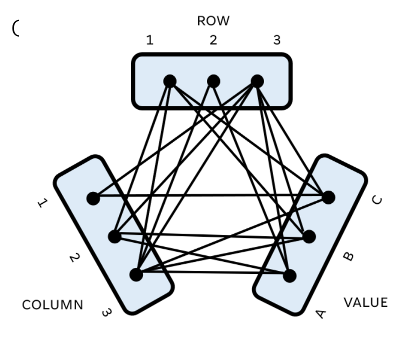


Figure 6. Graph Equivalent to L

If $r = s = t$ then LF is a latin square which can be completed iff G has a triangle partition.

Lemma: For graph $G = (V, E)$ with $|V| = n$, construct LF such that for $(r_i, c_j) \in E$, $LF(i, j) = 0$ and for $(r_i, c_j) \notin E$, $LF(i, j) = 1 + n + ((i + j) \text{mod } n)$. LF is a latin framework of size $(n, n, 2n)$ and can be extended to a Latin framework of size $(2n, 2n, 2n)$.

Given a tripartite graph G, if it is not uniform then no triangulation exists, else we apply the above lemmas to produce a latin framework of size $(2n, 2n, 2n)$ in polynomial time. This is a Latin square which can be completed if and only if G has a triangulation.

Latin Square is hard. $\square$

Latin Square $\leq_p$ Sudoku [4]

Finally we reach our star puzzle.

Lemma: Let S be a Sudoku problem with the following construction

$$S(i, j) = \begin{cases} 0 & \text{when } (i, j) \in S_l \\ ((i - 1 \text{ mod } n)n + \lfloor i - 1/n \rfloor + j - 1) \text{ mod } n^2 + 1 & \text{otherwise} \end{cases} \tag{1}$$

where $S_l = \{(i, j) | \lfloor i - 1/n \rfloor = 0 \text{ and } (j \text{ mod } n) = 1\}$. Then there exists an augmentation S' to complete the sudoku puzzle if and only if the square L such that $L(i, j/n) = (S'(i, j) - 1)/n + 1$ for all $(i, j) \in S_l$ is a Latin square.

Given partial Latin Square L, construct S as above and augment so $\forall (i, j)$ such that $L(i, j) \neq 0$ $S(i, n(j-1)+1) = n(L(i, j) - 1) + 1$. S after the augmentation can be solved if and only if L can be solved.

Sudoku is hard. $\square$

1		
2	3	

L

0	2	3	0	5	6	0	8	9
0	5	6	0	8	9	0	2	3
0	8	9	0	2	3	0	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

Lemma construction of S

1	2	3	0	5	6	0	8	9
4	5	6	7	8	9	0	2	3
0	8	9	0	2	3	0	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

S augmented

References

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006.

[2] Charles J. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8(1):25–30, 1984.

[3] Ian Holyer. The np-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981.

[4] Takayuki YATO and Takahiro SETA. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E86-A, 05 2003.