Sudoku is Hard

Eve Routledge

Sudoku: An Introduction

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.

Don't just take my word on it, instead follow this proof as we transform sudoku into a known 'difficult' problem (SAT) that has plagued computer scientists for decades.

We will transform sudoku to SAT 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 $S : i, j \rightarrow x$ 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 $a \bmod D = c \bmod D$, $b \bmod D = d \bmod D$, $S(a, b) \neq 0$ and $S(c, d) \neq 0$, then $S(a, b) \neq S(a, c)$

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

Def^n: A valid **Latin Square** puzzle is a function $L : i, j \rightarrow x$ 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, $b_1 \vee b_2 \vee ...b_i$. For this to be in conjugate normal form the formula must be a conjunction of clauses. A boolean expression is satisfiable if a truth assignment to all members of B 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

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**, $A \geq_p B$, is a transformation in $O(x^c)$ from problem B to A.

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

- P is the set of problems solved in polynomial time $O(x^c)$ 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 $O(x^c)$.

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 Sudoku$\geq_p$SAT we show sudoku is NP-complete and therfore 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 Sudoku$\geq_p$SAT 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 $\leq_p$ 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.

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.

3SAT is NP-Complete. □

Triangulated Tripartite Graph $\geq_p$ 3 SAT

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 $H_{3,p}$ defined as $V = \{(x_1, x_2, x_3) \in \mathbb{Z}^3 | x_1 + x_2 + x_3 = 0 \bmod p\}$, (x_1, x_2, x_3) and (y_1, y_2, y_3) are adjacent if there exists distinct i, j, k such that $x_i = y_i \bmod p$, $x_j = (y_j + 1) \bmod p$ and $x_k = (y_k - 1) \bmod p$.

Observe $H_{3,p}$ is a tripartite graph iff $p = 0 \bmod 3$.

$H_{3,p}$ is constructed such that there exists 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 glue graphs together by taking a set of vertices in G_1 and making them the 'same' as a set of vertices in G_2, such that the vertices sets are the same size.

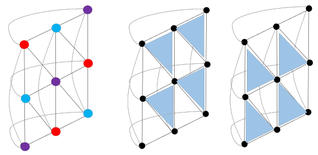

Figure 1. $H_{3,3}$, 3-colouring, true triangulation, false triangulation

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)

Lemma: Connecting two $H_{3,p}$ by two A-patches then removing the centre triangles from both graphs we get triangulations (T, F) or (F, T).

Lemma: Connecting x $H_{3,p}$ by x A-patches then removing the centre triangles from all graphs we get triangulations $\in \{(a_1, ..., a_x) | a_i = F \text{ and } \forall j \neq i \; a_j = T\}$.

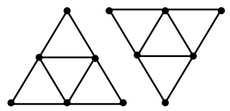

Figure 2. A-patch, B-patch

Using the above lemmas we can change a 3SAT boolean expression into a tripartite graoh that has a triangulation iff the boolean expression is satisfiable.

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 $G_{i,j}$ such that $l_{i,j} = b_i$ and a B-patch to $G_{i,j}$ such that $l_{i,j} = \neg b_i$. 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. Select p large enough to prevent patch overlap. $G = \{G_{b_i} \forall b_i \in B\} \cup \{G_{i,j} \forall c_j \in C \text{ and } l_{i,j} \in c_j\}$

If a triangulation T of G exists then:

- G_{b_i} has a true or false triangulation.
- If $l_{i,j} = b_i$, $G_{i,j} = T$ when $G_{b_i} = F$ but when $G_{b_i} = T$ $G_{i,j}$ can be a true or false triangulation.
- If $l_{i,j} = \neg b_i$, $G_{i,j} = T$ when $G_{b_i} = T$ but when $G_{b_i} = T$ $G_{i,j}$ can be a true or false triangulation.
- Exactly one of $G_{i,1}, G_{i,2}$ or $G_{i,3}$ have a false triangualtion and the other have a true triangulations.

These statements come from the above lemmas and allow a 3SAT formula to be converted to the problem of triangulating a tripartite graph. Conversely if we can find a triangulation this will give us a truth assignment which can be determined from the G_{b_i} portions of the graph. If G_{b_i} is a true triangulation $b_i = T$ in our truth assignment. □

References

[1] Elineo Hoexum. Revisiting the proof of the complexity of the sudoku puzzle. 2020.

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

Latin Square $\geq_p$ Triangulated Tripartite Graph

Observe completing a partial Latin square is equivalent to triangulating tripartite graph, we map the Latin square to this through the following: given tripartite graph $G = (V, E)$ label vertices in V_1 with distinct lables $\{r_1, ...r_n\}$, label vertices in V_2 with distinct lables $\{c_1, ...c_n\}$ and label vertices in V_3 with distinct lables $\{e_1, ...e_n\}$. Add edges such that:

- If $L(i, j) = 0$ then add the edge (r_i, c_j)
- If for all $i \in [0, ..., n]$ and constant j, $L(i, j) \neq k$ then add the edge (r_i, e_k)
- If for all $j \in [0, ..., n]$ and constant i, $L(i, j) \neq k$ then add the edge (c_j, e_k)

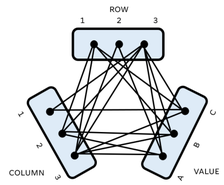

Figure 3. Tripartite Graph equivalent to partial L

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

Let us show every uniform tripartite graph is the above formulation of a Latin square.

Def^n: A Latin framework LF for tripartite graph G, 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 $LF(i, j) =$empty else $LF(i, j) = k$ $k \in [1, ..., t]$
- 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 $LF(i, j) \neq k$

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 $|V| = n$, there's a Latin framework of size $(n, n, 2n)$.

Define LF an $n \times n$ array. For $(r_i, c_j) \in E$ $LF(i, j) = 0$ else $LF(i, j) = 1 + n + ((i + j) \bmod n)$.$LF$ is a latin framework.

Lemma: Latin framework L (r, s, t) for uniform tripartite graph G. $R(k) =$ the number of times k appears in L plus half $|e_k|$. Whenever $R(k) \geq r + s - t$ for $1 \leq k \leq t$, L can be extended to $(r, s + 1, t)$ to give L' in which $R'(k) \geq r + s + 1 - t$ for all $1 \leq k \leq t$.

Lemma: Latin framework of size (r, s, s) can be extended to a Latin framework size (s, s, s).

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

Sudoku $\geq_p$ Latin Square

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

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

where $S_l = \{(i, j) | \lfloor i - 1/n \rfloor = 0 \text{ and } (j \bmod n) = 1\}$. 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.

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

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. □

https://www.example.com

eve.routledge@durham.ac.uk