SURVEY

Finding an efficient method to solve SuDoku puzzles is:

1: A waste of time
2: A decent spare time activity
3: A fundamental problem in computer science
SURVEY

Finding an efficient method to solve SuDoku puzzles is:

1: A waste of time
2: A decent spare time activity
3: A fundamental problem in computer science
Does every problem have efficient algorithms?

Halting Problem: Given program code, output whether program halts or not.

Theorem [Godel]: Halting cannot be solved by any algorithm.
Theorem: Integer Equations cannot be solved by any algorithm.

What about problems that have algorithms? Must they have efficient algorithms?

Theorem: There are problems that can be solved in exponential time, but not in polynomial time.

OK, but what about Set Cover, Vertex Cover, Shortest Spanning Path - all have brute force algorithms, but do they have efficient algorithms?
Decision Problems

**Decision problem**: Problems with “yes” or “no” answers.

- Does a given set system have a set cover of size at most $k$?
- Does a given graph have a vertex cover of size at most $k$?
- Does a number have a non-trivial factorization?
- Does a given graph have an MST of cost at most $k$?
- Does a given flow network have a min-cut of capacity at most $k$?
- Does a given sudoku problem have a solution?

**Polynomial time.** Algorithm $A$ runs in poly-time if for every string $x$, $A(x)$ terminates in at most $p(|x|)$ "steps", where $p$ is some polynomial.

↑

length of $x$

$P$: The class of decision problems that can be solved in polynomial time.

**PRIMES**: $X = \{2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots\}$. Is input a prime?

**Theorem** [Agrawal-Kayal-Saxena, 2002] PRIMES is in $P$. 

Certification algorithm intuition.

Certifier doesn't determine whether answer is “yes” on its own; rather, it checks a proposed proof $t$ that answer is “yes”.

**Def.** Algorithm $C(x, t)$ is a certifier for problem $X$ if for every string $x$, the answer is “yes” iff there exists a string $t$ such that $C(x, t) = \text{yes}$.

**NP.** Decision problems for which there exists a poly-time certifier.

- $C(x, t)$ is a poly-time algorithm and $|t| \leq p(|x|)$ for some polynomial $p$.

**Remark.** NP stands for nondeterministic polynomial-time.
Certifiers and Certificates: Composite

COMPOSITES. Given an integer \( x \), is \( x \) composite?

Certificate. A nontrivial factor \( t \) of \( x \). Note that such a certificate exists iff \( x \) is composite. Moreover \( |t| \leq |s| \).

Certifier.

```java
boolean C(x, t) {
    if (t = 1 or t = x)
        return false
    else if (x is a multiple of t)
        return true
    else
        return false
}
```

Instance. \( x = 437,669 \).
Certificate. \( t = 541 \) or 809. \( 437,669 = 541 \times 809 \)

Conclusion. COMPOSITES is in NP.
Certifiers and Certificates: 3-Satisfiability

3SAT. Given a 3-CNF formula, is there a satisfying assignment?

Certificate. An assignment of truth values to the $n$ boolean variables.

Certifier. Check that each clause has at least one true literal.

Ex.

\[
(\overline{x}_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x}_2 \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (\overline{x}_1 \lor \overline{x}_3 \lor \overline{x}_4)
\]

instance $s$

\[
x_1 = 1, \ x_2 = 1, \ x_3 = 0, \ x_4 = 1
\]

certificate $t$

Conclusion. 3SAT is in NP.
Certifiers and Certificates: Hamiltonian Cycle

HAM-CYCLE. Given an undirected graph $G = (V, E)$, does there exist a simple cycle $C$ that visits every node?

Certificate. A permutation of the $n$ nodes.

Certifier. Check that the permutation contains each node in $V$ exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

Conclusion. HAM-CYCLE is in NP.
Certifiers and Certificates: Min-Cut

MIN-CUT. Given a flow network, and a number k, does there exist a min-cut of capacity at most k?

Certificate. A min-cut T.

Certifier. Check that the capacity of the min-cut is at most T.

Conclusion. MIN-CUT is in NP.
**Certifiers and Certificates: Min-Cut**

**MIN-CUT.** Given a flow network, and a number k, does there exist a min-cut of capacity at most k?

**Certificate.** The empty string.

**Certifier.** Compute the min-cut of the graph and check whether its capacity is at most k.

**Conclusion.** MIN-CUT is in NP.
Examples of NP Problems

Eg: Does a given set system have a set cover of size at most \( k \)?
   Certificate: A set cover of size at most \( k \)

Does a given graph have a vertex cover of size at most \( k \)?
   Certificate: A vertex cover of size at most \( k \)

Does a number have a non-trivial factorization?
   Certificate: A non-trivial factorization

Does a given graph have an MST of cost at most \( k \)?
   Certificate: An MST of cost at most \( k \)

Does a given flow network have a min-cut of capacity at most \( k \)?
   Certificate: A min-cut of capacity at most \( k \)

Does a given sudoku problem have a solution?
   Certificate: A valid solution.
P. Decision problems for which there is a *poly-time algorithm*.

**EXP.** Decision problems for which there is an *exponential-time algorithm*.

NP. Decision problems for which there is a *poly-time certifier*.

Claim. \( P \subseteq NP \).

Pf. Consider any problem \( X \) in \( P \).

- By definition, there exists a *poly-time algorithm* \( A(x) \) that solves \( X \).
- **Certificate:** \( t = \) empty string, **certifier** \( C(x, t) = A(x) \).

Claim. \( NP \subseteq EXP \).

Pf. Consider any problem \( X \) in \( NP \).

- By definition, there exists a *poly-time certifier* \( C(x, t) \) for \( X \).
- To solve input \( x \), run \( C(x, t) \) on all strings \( t \) with \( |t| \leq p(|x|) \) (running time of \( C \)).
- **Return** \( yes \), if \( C(x, t) \) returns \( yes \) for any of these.
The Main Question: P Versus NP

Does $P = NP$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
- Is the decision problem as easy as the certification problem?
- Clay $1$ million prize.

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...
If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...
NP-Completeness

Punchline: If you find a way to solve sudoku in polynomial time, you will solve factoring in polynomial time!
NP-Completeness

Punchline: If you find a way to solve sudoku in polynomial time, you will solve set cover in polynomial time!
NP-Completeness

Punchline: If you find a way to solve sudoku in polynomial time, you will solve SAT in polynomial time!
NP-Completeness

Punchline: If you find a way to solve sudoku in polynomial time, you will solve all machine learning problems in polynomial time!
NP-Completeness

Punchline: If you find a way to solve sudoku in polynomial time, you will solve every problem in NP in polynomial time!
NP-Completeness

**Def.** Problem X polynomial reduces to problem Y \((X \leq_p Y)\) if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to subroutine that solves problem Y.

**NP-complete Problem.** A problem Y in NP with the property that for every problem X in NP, \(X \leq_p Y\).

**Theorem.** Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff \(P = NP\).

**Pf. \(\Leftarrow\)** If \(P = NP\) then Y can be solved in poly-time since Y is in NP.

**Pf. \(\Rightarrow\)** Suppose Y can be solved in poly-time.

- Let X be any problem in NP. Since \(X \leq_p Y\), we can solve X in poly-time. This implies \(NP \subseteq P\).
- We already know \(P \subseteq NP\). Thus \(P = NP\). ▪

**Fundamental question.** Do there exist "natural" NP-complete problems?
Program Satisfiability

PROGRAM-SAT. Given a line program on inputs $x=x_1,x_2,...,x_n$ is there a way to set the inputs so that the output is 1?

\[
\begin{align*}
    l_1 &= x_1 \text{ AND } x_2; \\
    l_2 &= x_3 \text{ OR } x_5; \\
    l_3 &= \text{NOT } x_6 \text{ AND } x_8; \\
    l_4 &= l_1 \text{ XOR } l_3; \\
    l_5 &= l_2 \text{ AND } x_4; \\
    l_6 &= \text{NOT } l_4 \text{ OR } l_2; \\
    &\vdots \\
    l_{m-2} &= l_{17} \text{ AND } l_{25}; \\
    l_{m-1} &= x_1 \text{ XOR } x_2; \\
    l_m &= x_1 \text{ XOR } l_{m-2}; \\
    \text{OUTPUT } l_m
\end{align*}
\]
The "First" NP-Complete Problem

**Theorem.** PROGRAM-SAT is NP-complete. [Cook 1971, Levin 1973]

**Pf.** (sketch)
- Any polynomial time algorithm can be compiled into a poly-size program.
- If problem X has poly-time certifier $C(x, t)$, to solve X, need to know if there exists a certificate $t$ such that $C(x, t) = \text{yes}$.
- Let $K(t)$ be poly-size program computing $C(x, t)$
- Program $K(t)$ is satisfiable iff $X(x) = \text{yes}$. 
Recipe to establish NP-completeness of problem Y.

- Step 1. Show that Y is in NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that $X \leq_P Y$.

Justification. If $X$ is an NP-complete problem, and $Y$ is a problem in NP with the property that $X \leq_P Y$ then $Y$ is NP-complete.

Pf. Let $W$ be any problem in NP. Then $W \leq_P X \leq_P Y$.

- By transitivity, $W \leq_P Y$.
- Hence $Y$ is NP-complete. ▪

by assumption
by definition of NP-complete
Theorem. 3-SAT is NP-complete.

Pf. Suffices to show that PROGRAM-SAT $\leq_P$ 3-SAT since 3-SAT is in NP.

- Let $K$ be any line program.
- Create a 3-SAT variable $l_i$ for each line $i$.
- Make variables compute correct values at each node:
  - $l_i = l_4 \text{ AND } x_5$ add 4 clauses: $(l_i \text{ OR not } l_4 \text{ OR not } x_5) \text{ AND (} l_i \text{ OR not } l_4 \text{ OR } x_5)$ AND $(l_i \text{ OR } l_4 \text{ OR not } x_5) \text{ AND (} not l_i \text{ OR } l_4 \text{ OR } x_5)$
- 3SAT formula is satisfiable if and only if $K$ is satisfiable.
**Observation.** All problems below are NP-complete and polynomial reduce to one another!
More NP-Complete Computational Problems

Aerospace engineering: optimal mesh partitioning for finite elements.
Biology: protein folding.
Chemical engineering: heat exchanger network synthesis.
Civil engineering: equilibrium of urban traffic flow.
Economics: computation of arbitrage in financial markets with friction.
Electrical engineering: VLSI layout.
Environmental engineering: optimal placement of contaminant sensors.
Financial engineering: find minimum risk portfolio of given return.
Game theory: find Nash equilibrium that maximizes social welfare.
Genomics: phylogeny reconstruction.
Mechanical engineering: structure of turbulence in sheared flows.
Medicine: reconstructing 3-D shape from biplane angiocardio gram.
Operations research: optimal resource allocation.
Physics: partition function of 3-D Ising model in statistical mechanics.
Politics: Shapley-Shubik voting power.
Pop culture: Minesweeper consistency.
Statistics: optimal experimental design.