NP-Hardness Proofs With Realistic Computers Instead of Turing Machines: Towards Making Theory of Computation Course More Understandable and Relevant

Olga Kosheleva\textsuperscript{1} and Vladik Kreinovich\textsuperscript{2}

Departments of \textsuperscript{1}Teacher Education and \textsuperscript{2}Computer Science
University of Texas at El Paso, El Paso, TX 79968, USA
olgak@utep.edu, vladik@utep.edu
1. NP-Hardness Proofs Are Important

- In many application areas, certain problems are known to be NP-hard (= provably computationally intractable).
- Knowing that a general problem is NP-hard helps the researchers to concentrate on easier-to-solve problems:
  - to find a practically useful easier-to-solve subclass of problems, or
  - to replace the original problem with a relaxed easier-to-solve problem.
- For example, we may only want an approximate solution, or an answer which is correct w/high probability.
- It is important to make sure that the new problem is indeed easier-to-solve.
- Thus, it is desirable that the students learn how to prove NP-hardness or different problems.
2. Usual Proofs of NP-Hardness

- A historically first problem proven to be NP-hard is *propositional satisfiability*.
- This problem is about *propositional formulas*, i.e., expressions $F$ like $(x_1 \& x_2) \lor (x_2 \& \neg x_3)$ obtained:
  - from propositional (“yes”-“no”) variables $x_1, \ldots, x_n$,
  - by using “and” ($\&$), “or” ($\lor$), and “not” ($\neg$).
- We are given a propositional formula $F$, we must find values $x_1, \ldots, x_n$ that make it true.
- The usual NP-hardness proof uses *Turing machines*, a simple theoretical computer designed in 1936.
- A Turing machine is, in effect, a tape recorder with a simple controller and a potentially extendable tape.
- For example, in the Turing machine, there is no immediate access to a memory cell at a given location.
3. Proofs of NP-Hardness (cont-d)

- The only way to get to a cell #1,000,000 is to go from cell #0 to cell #1, to cell #2, . . . , to cell #1,000,000.

- It is amazing to learn that complex computations can be performed on such a primitive computer.

- However, when it comes to proving that no efficient algorithm exists:
  
  – the fact that, for some problem, no efficient solutions are possible on a Turing machine
  
  – is not a very convincing argument that this is impossible on (more complex) real computers.

- Yes, there are proofs that Turing machines are sufficient for proving NP-hardness.

- However, these proofs are beyond the scope of most textbooks.
4. Pedagogical Problem and What We Do About It

- As we mentioned, for students, Turing-machine-based NP-hardness proofs are not convincing at all.
- We propose a new version of the proof of NP-hardness of propositional satisfiability.
- This proof that uses a much more realistic (and general) model of a computer than Turing machine.
- This proof is somewhat more complex than the Turing-machine-based proofs.
- However, our model (and hence this proof) is closer to the actual computers and is, thus, easier to understand.
5. What Problems We Are Solving: Examples

- In mathematics, we are given a statement $x$ and we want to find the proof $y$ of either $x$ or $\neg x$.
- Once we have a detailed proof $y$, it is easy to check its correctness, but inventing a proof is hard.
- A proof cannot be too long: it must be checkable.
- In physics, we have observations $x$, and we want to find a law $y$ that describes them.
- Once we have $y$ we can easily check whether it fits $x$, but coming up with $y$ is often difficult.
- A law cannot be too long: otherwise, we can take the data as the law.
- In engineering, we have a specification $x$, and we need to find a design $y$ that satisfies $x$. 
6. What Problems We Are Solving: General Description

• In general:
  – we have a string $x$, and
  – we need to find $y$ s.t. $C(x, y)$ and $\text{len}(y) \leq P_\ell(\text{len}(x))$.

• Here, $C(x, y)$ is a feasible property, i.e., a property that can be checked feasibly (in polynomial time).

• In such problems:
  – once we have a guess $y$,
  – we can check its correctness in polynomial time.

• “Computations” allowing guesses are known as non-deterministic.

• Thus, such problems are called Non-deterministic Polynomial (NP).
7. What Is NP-Hard: Reminder

- Ideally, we would like to call a problem *hard* if it cannot be solved by a feasible (polynomial-time) algorithm.

- Alas, for neither of the problems from NP, we can prove that this problem is hard in this sense.

- What we do know is that some problems are *harder* than others in the following sense:
  - every instance of a problem \( A \)
  - can be reduced to an appropriate instance of the problem \( B \).

- A problem is called *NP-hard* if every problem from NP can be reduced to it.

- In other words, a problem is NP-hard if it is harder than all other problems from the class NP.
8. Proof that Satisfiability Is NP-Hard: Idea

• We have an instance of an NP problem: given $x$ find $y$ for which $C(x, y)$ is true and $\text{len}(y) \leq P_\ell(\text{len}(x))$.

• We want to reduce it to propositional satisfiability.

• We start with a computational device that, given a string $x$ of length $\text{len}(x) = n$ and $y$, checks $C(x, y)$.

• Computing $C$ requires polynomial time $T \leq P(n)$.

• During this time, only cells at distance $\leq R = c \cdot T$ from the origin can influence the result.

• Let $\Delta V$ be the smallest cell volume.

• Within the sphere of volume $V = \frac{4}{3} \cdot \pi \cdot R^3 \sim T^3$, there are $\leq \frac{V}{\Delta V} \sim T^3$ cells, fewer than $\leq \text{const} \cdot (P(n))^3$.

• So, we have no more than polynomially many cells.
9. Proof that Satisfiability Is NP-Hard (cont-d)

- Let $\Delta t$ be a time quantum.
- The state $S_{i,t+1}$ cell $i$ at moment $(t+1) \cdot \Delta t$ can only be influenced by states $S_{j,t}$ of cells at distance $\leq r = c \cdot \Delta t$.
- In this vicinity, there are $\leq N_{\text{neigh}} = \frac{4}{3} \cdot \pi \cdot \frac{r^3}{\Delta V}$ cells; this number does not depend on the inputs size $n$:
  \[ S_{i,t+1} = f_{i,t}(S_{i,t}, S_{j,t}, \ldots (\leq N_{\text{neigh}} \text{ terms})). \]
- Let $S$ be the largest number of states of each cell.
- We can describe each state as 0, 1, 2, \ldots
- Then we need $B \overset{\text{def}}{=} \lceil \log_2(S) \rceil$ bits $s_{i,b,t}$, $1 \leq b \leq B$, to describe each state $S_{i,t}$, so:
  \[ s_{i,b,t+1} = f_{i,t}(s_{i,1,t}, \ldots, s_{i,B,t}, s_{j,1,t}, \ldots, s_{j,B,t}, \ldots). \]
- We can then use a truth table to transform each such equation to a propositional formula $F_{i,b,t}$. 
10. Proof that Satisfiability Is NP-Hard (final steps)

- For each cell $i$, bit $b$, and moment of time $t$, the fact that $s_{i,b,t+1}$ is computed correctly can be described as
  \[
  s_{i,b,t+1} = f_{i,t}(s_{i,1,t}, \ldots, s_{i,B,t}, s_{j,1,t}, \ldots, s_{j,B,t}, \ldots).
  \]

- We have shown that this property can be described by a propositional formulas $F_{i,b,t}$.

- By combining all these formulas, we get a long formula
  \[
  F_{\text{long}} \overset{\text{def}}{=} F_{1,1,1} \& F_{1,2,1} \& \ldots \& F_{i,b,t} \& \ldots
  \]

- Meaning of $F_{\text{long}}$: that $C(x, y)$ was checked correctly.

- We add the formulas describing that the input was $x$ and that the output of checking $C(x, y)$ was “true”.

- The resulting propositional formula holds if and only if there exists $y$ for which $C(x, y)$ is satisfied.

- Reduction is proven, so satisfiability is indeed NP-hard.
11. Acknowledgment

This work was supported in part

- by the National Science Foundation grants HRD-0734825 (Cyber-ShARE Center) and DUE-0926721, and

- by Grant 1 T36 GM078000-01 from the National Institutes of Health.