Designing, Understanding, and Analyzing Unconventional Computation

Vladik Kreinovich

Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@utep.edu

1. Main Problem: Reminder and Natural Idea

- *Problem:* computations are often too slow.
- *Traditional approaches:*
  - design faster super-computers (*hardware*);
  - design faster *algorithms*.
- *Limitations of the traditional approaches:*
  - *re hardware:* we use the same physical processes as before;
  - *re algorithms:* we solve the same *exact* problem as before.
- *Possible new approach:* use unconventional processes:
  - unconventional physical processes, or
  - unconventional biological processes.

2. Which Problems Are Feasible: Brief Reminder

- *In theoretical computer science:* researchers usually distinguish between
  - problems that can be solved in polynomial time, i.e., in time $\le P(n)$ where n is input length, and
  - problems that require more computation time.
- *Terminology:*
  - problems solvable in polynomial time are usually called *feasible*,
  - while others are called *intractable*.
- *Warning:* this association is not perfect.
- *Example:* an algorithm that requires $10^{100} \cdot n$ steps is
  - polynomial time, but
  - not practically feasible.

3. 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.

4. 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 $\text{len}(y) \le P_\ell(\text{len}(x))$.
- 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).

5. 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.

6. Propositional Satisfiability: Historically First NP-Hard Problem

- In many applications areas, certain problems are known to be NP-hard (= provably computationally intractable).
- A historically first problem proven to be NP-hard is *propositional satisfiability*.
- This problem is about *propositional formulas*, i.e., expressions F like $(x_1 \,\&\, x_2) \vee (x_2 \,\&\, \neg x_3)$ obtained:
  - from propositional ("yes"-"no") variables $x_1, \ldots, x_n$,
  - by using "and" (&), "or" ($\vee$), and "not" ($\neg$).
- We are given a propositional formula F, we must find values $x_1, \ldots, x_n$ that make it true.

7. 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 $\text{len}(y) \le P_\ell(\text{len}(x))$.
- 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 $T \le P(n)$.
- During this time, only cells at distance $\le R = c \cdot T$ from the origin can influence the result.
- Let ΔV be the smallest cell volume.
- Within the sphere of volume $V = \dfrac{4}{3} \cdot \pi \cdot R^3 \sim T^3$, there are $\le \dfrac{V}{\Delta V} \sim T^3$ cells, fewer than $\le \text{const} \cdot (P(n))^3$.
- So, we have no more than polynomially many cells.

8. Proof that Satisfiability Is NP-Hard (cont-d)

- Let Δt 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 $\le r = c\cdot\Delta t$.
- In this vicinity, there are $\le N_{\text{neighb}} = \frac{4}{3}\cdot\pi\cdot\frac{r^3}{\Delta V}$ cells; this number does not depend on the inputs size n:

$$S_{i,t+1} = f_{i,t}(S_{i,t}, S_{j,t}, \ldots (\le N_{neighb} \text{ terms})).$$

- Let S be the largest number of states of each cell.
- We can describe each state as 0, 1, 2, ...
- Then we need $B \stackrel{\text{def}}{=} \lceil \log_2(S) \rceil$ bits $s_{i,b,t}$, $1 \le b \le B$, to describe each state $S_{i,t}$, so:

$$s_{i,b,t+1} = f_{i,b,t}(s_{i,1,t}, \ldots, s_{i,B,t}, s_{j,1,t}, \ldots, s_{j,B,t}, \ldots).$$

- We can then use a truth table to transform each such equation to a propositional formula $F_{i,b,t}$.

9. 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

$$F_{\text{long}} \stackrel{\text{def}}{=} F_{1,1,1} \,\&\, F_{1,2,1} \,\&\, \ldots \,\&\, F_{i,b,t} \,\&\, \ldots$$

- Meaning of F_{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.

10. Is Speed Up Possible?

- *Problem:* some problems are intractable – i.e., require algorithms which are too slow (intractable).
- *Clarification:* they are slow when we use the physical processes which are currently used in computers.
- *Natural idea:* use new physical processes, processes that have not been used in modern computers.
- *Question:* is it possible to make computations drastically faster?
- *Reformulation:* is it possible to make intractable problems feasible?
- *This may happen:* if a physical process provides a super-polynomial (= faster than polynomial) speed-up.

11. Natural Ideas

- In our proof that satisfiability is NP-hard, we used two physical assumptions:
  - that space is Euclidean, so the volume of a sphere of radius R is $\sim R^3$, and
  - that all the speeds are limited by the speed of light.
- *Natural ideas:*
  - take into account that actual space is not Euclidean;
  - use hypothetical faster-than-light (acausal) processes.

12. Parallelization: Reminder

- If we accumulate a lot of parallel processors, maybe we solve exponential-time problems in polynomial time?
- *Result:* parallelism cannot reduce the computation time T that drastically.
- During the parallel computation time T_p, we can only access computers within a sphere of radius $R = c \cdot T_p$.
- Within this sphere of volume $V = \frac{4}{3} \cdot \pi \cdot R^3 \sim T_p^3$, we can fit $\le V/\Delta V \sim T_p^3$ processors of given size ΔV.
- All these processors can perform $T \le \frac{T_p}{\Delta t} \cdot \text{const} \cdot T_p^3 = C \cdot T_p^4$ computational steps.
- So, if a computation requires T sequential steps, we need $T_p \ge C \cdot T^{1/4}$ steps to perform it in parallel.

13. Parallelization in Curved Space-Time

- *Observation:* the above lower bound on parallel computation time depends on the formula $V(R) = \frac{4}{3} \cdot \pi \cdot R^3$.
- *Known:* this formula only holds in Euclidean geometry.
- *Idea:* since the actual space-time is curved (= not Euclidean), we may get faster parallel computations.
- *Known:* in Lobachevsky space,

$$V(R) = 2\pi k^3 \cdot \left(\sinh\left(\frac{R}{k}\right) \cdot \cosh\left(\frac{R}{k}\right) - \frac{R}{k} \right) \sim \exp\left(\frac{2}{k} \cdot R\right).$$

- *Corollary:* we can fit exponentially many processors into a sphere of radius $R = c \cdot T_p$.
- *Conclusion:* in Lobachevsky space, parallelization can reduce exponential time $T = 2^n$ to linear time $T_p \sim n$.
- *Lobachevsky's idea:* by measuring $V(R)$, we can speed up computation of $\sinh(x)$.

14. Parallelization in Curved Space-Time (cont-d)

- *Good news:* in Lobachevsky (constant curvature) space, parallelization speeds up computations.
- *Problem:* actual space-time is more complex.
- *Good news:* there exist more realistic space-time models with the same property.
- *Known:* there is no way to escape from a black hole.
- *Known:* as the matter collapses, the escape throat gets narrower.
- There exist "almost" black hole models, with a throat so narrow that they look like elementary particles.
- *Known hypothesis:* particles are such "almost" black holes, entering into other "universes".

15. Parallelization in Curved Space-Time (cont-d)

- *Assumption:* particles are such "almost" black holes, entering into other "universes".
- Let us show how this can help us solve NP-hard problems.
- *Example:* propositional satisfiability SAT:
  - given a propositional formula $F(x_1, \ldots, x_n)$,
  - find the values of the variables $x_1, \ldots, x_n$ that make $F(x)$ true.
- To find $x = (x_1, \ldots, x_n)$, $x_i \in \{0, 1\}$, s.t. $F(x)$, we:
  - find two particles (and corr. worlds);
  - ask World 1 to search for $x = (0, x_2, \ldots, x_n)$ s.t. $F(x)$;
  - ask World 2 to search for $x = (1, x_2, \ldots, x_n)$ s.t. $F(x)$.
- Each of these worlds does the same split w.r.t. x_2, etc.; in time $2n$ ($\ll 2^n$), we get an answer back.

16. Acausal Processes: Reminder

- *Known fact:* several physical theories have led to micro- and macro-causality violations, i.e., going back in time.
- *Feynman:* positrons are electrons going back in time.
- *Mainstreaming:* K. Thorne's Physical Reviews papers.
- *General relativity:* space-time generated by a massive fast-rotating cylinder contains a closed timelike curve.
- *String theory*: interactions between string-like particles sometime lead to the possibility to influence the past.
- *Cosmology:*
  - a short initial period of exponentially fast growth ("inflation")
  - can lead to a causal anomaly.

17. Acausal Processes: Analysis

- *Paradox of causality violation:*
  - a time traveler goes into the past and
  - kills his father before he himself was conceived.
- *Solution:* since the time traveler was born, some unexpected event prevented him from killing his father.
- The time traveler takes care of all such probable events.
- *But:* we cannot avoid all events with small probability.
- *Example:* a meteor can fall on the traveler's head and prevent him from killing his father.
- *Conclusion:* time travel may be possible.
- *How to use it for computations:* a computer computes and send the result back in time, to us now.

18. Using Acausal Processes for Computations

- *Alternative algorithm* for solving SAT:
  - generate n random bits $x_1, \ldots, x_n$ and check whether they satisfy a given formula $F(x_1, \ldots, x_n)$;
  - if not, launch a time machine that is set up to implement a low-probability event.
- *Analysis:* nature has two choices:
  - generates n variables which satisfy the given formula (probability 2^{-n}),
  - time machine is used, triggering an event with probability p_0.
- If $2^{-n} \gg p_0$, then the first event is much more probable.
- So, the solution to the satisfiability problem will actually be generated.
- *Interesting:* there is no actual time travel.

19. Quantum Computing

- *Question* (reminder): find physical processes that would make computations drastically faster.
- *Most active* research in this direction – quantum computing.
- *Fact:* quantum processes can speed up computations.
- *Example:* Grover's algorithm searches in an un-sorted list of size N in time $\sqrt{N}$.
- *Application:* to problems that can be solved by $N = 2^n$ time exhaustive search.
- *Example:* SAT – given a propositional formula $F(x)$, find $x = (x_1, \ldots, x_n)$ s.t. $F(x)$ holds.
- *Exhaustive search:* try all 2^n possible combinations of $x_i \in \{false, true\}$.

20. Quantum Computing: Limitations

- *Reminder:* SAT under quantum computing.
- *Grover's algorithm:* reduces the computation time from $N = 2^n$ to
$$\sqrt{N} = \sqrt{2^n} = 2^{n/2}.$$
- *Limitation:* this is still a polynomial-time speed-up:
  - let $T_c(n)$ be non-quantum time, then quantum time is $T_q(n) = \sqrt{T_c(n)}$;
  - when $T_q(n)$ is polynomial, so is $T_c(n) = T_q^2(n)$:-)
- *Fact:* some known quantum algorithms are exponentially faster than the best known non-quantum ones.
- *Example:* Shor's algorithm for factoring large integers.
- *Limitation:* it is not clear whether a similar fast non-quantum algorithm is possible.
- The only *proven* quantum speed-ups are polynomial.

21. Idea: Explicit Use of Kolmogorov Complexity

- *Fact:* it's often difficult to describe biological processes.
- *Idea* (M. Gell-Mann): physical equations should include terms explicitly depending on complexity.
- *Natural formalization:* Kolmogorov complexity

$$K(x) \stackrel{\text{def}}{=} \min\{\text{len}(p) : p \text{ generates } x\}.$$

- *Conclusion:* by observing physical and biological processes, we can measure the value $K(x)$.
- *Observation:* $K(x)$ is not algorithmically computable.
- *Known results:* ability to get non-computable values can speed up computations.
- *Conclusion:* by observing biological processes, we can
  - determine $K(x)$, and thus
  - speed up computations.