# Algorithmic Persuasion with No Externalities

Shaddin Dughmi and Haifeng Xu

**Discussant:** Adrian Vetta

**Trottier Fellow in Science and Public Policy**

School of Computer Science
and
Department of Mathematics and Statistics

*McGill University*

# The Central Paradigm of Computer Science

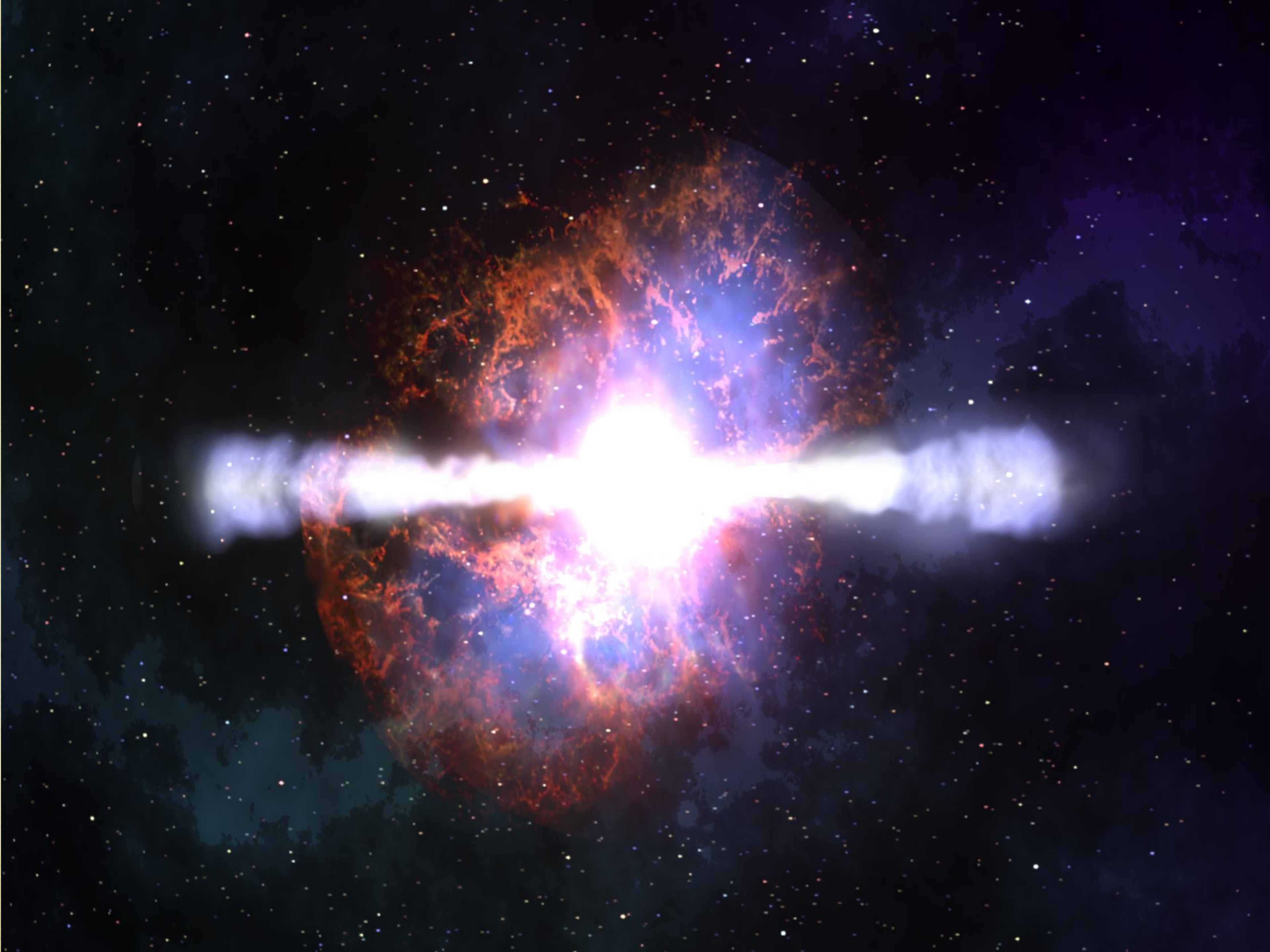
- The central <u>paradigm</u> in computer science is that an algorithm $A$ is **good** if:
  - $A$ runs in *polynomial time* in the input size $n$.
- That is, $A$ runs in time $T(n) \;=\; O(n^k)$ for some constant number $k$.

*e.g.* $T(n) = 100n + 55$ $\qquad$ $T(n) = \frac{1}{2}n^2 + 999 \log n$

$$T(n) = 6n^7 - n^3 + 899890n^2 - \sqrt{n}$$

- An algorithm is **bad** if it runs in *exponential time*.

*e.g.* $T(n) = 2^n + 100n^5$ $\qquad$ $T(n) = 1.00000001^n - n^3 - n$




- An agent wants to discover its *preference ordering* over $n$ outcomes.
  - This is simply the problem of sorting $n$ numbers.

<u>A Good Algorithm</u>: MergeSort runs in time $O(n \cdot \log n)$

<u>A Bad Algorithm</u>: ExhaustiveSearch runs in time $O(n \cdot n!) \gg 2^n$

- The functionality of our economic system is based on this paradigm!
  - **Public-Key Cryptography**: *Message senders and recipients* have good algorithms to encrypt and decrypt.
  - An *eavesdropper* has a bad algorithm to decript [prime factorization].

# An Equivalent Characterization

- This central <u>paradigm</u> has an equivalent formulation:
  - $\mathcal{A}$ runs in **polynomial time** in the input size $n$.

$$\equiv$$

  - The input sizes that $\mathcal{A}$ can solve, *in a fixed amount T of time*, **scales multiplicatively** with increasing computational power.

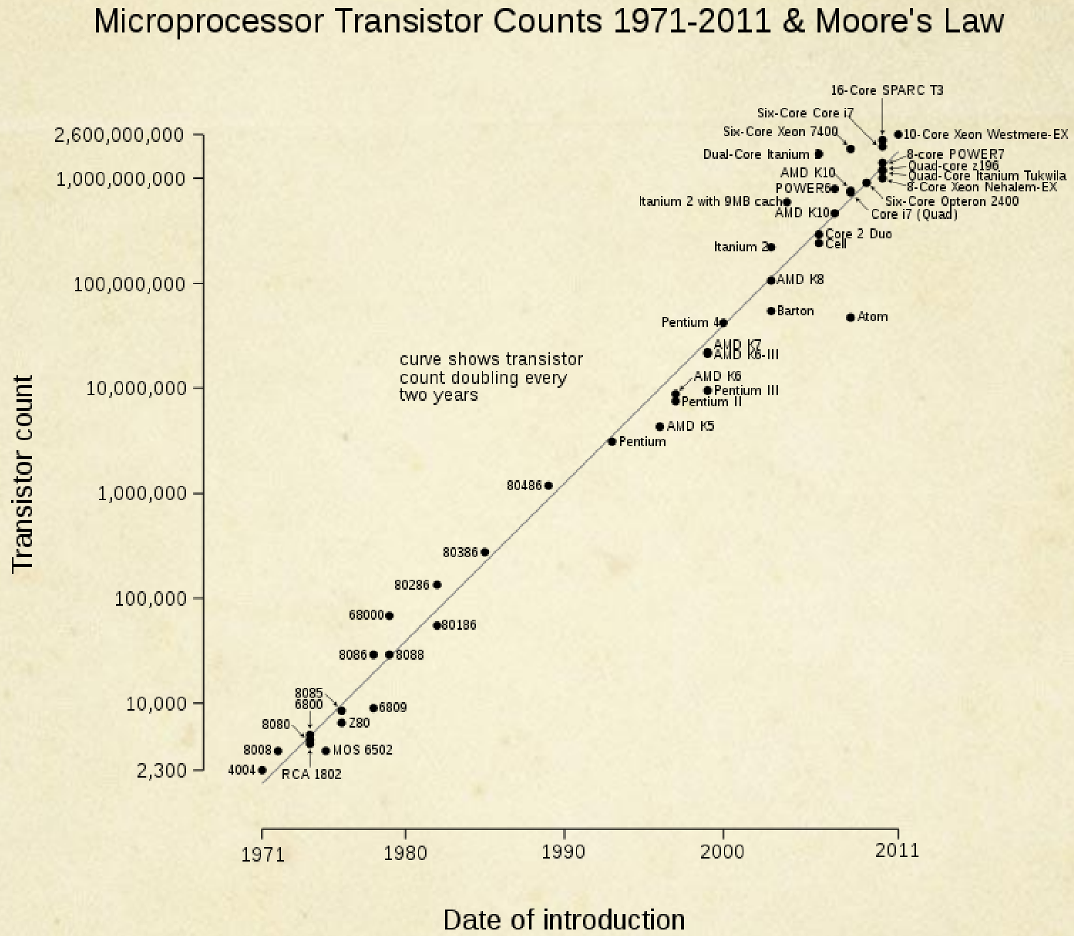
# Multiplicative Scalability

- An algorithm *A* is **good** if the input sizes it can solve, *in a fixed amount T of time*, scales <u>multiplicatively</u> with increasing **computational power**.

**Input Sizes solved in Time T**

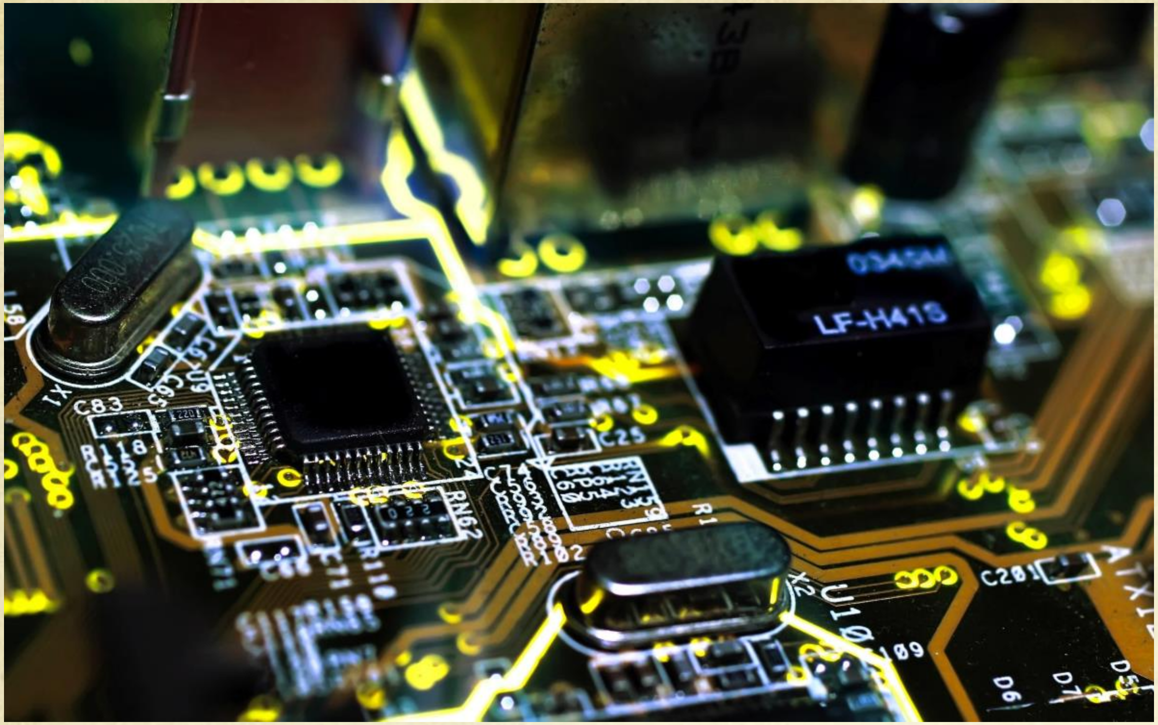
| Runtime of Algorithm | | $\text{Power} = 1$ | $\text{Power} = 2$ |
|---|---|---|---|
| | $n$ | $T$ | $2T$ |
| | $n^2$ | $\sqrt{T}$ | $\sqrt{2} \cdot \sqrt{T}$ |
| | $2^n$ | $\log T$ | $1 + \log T$ |

# Moore's Law




**Moore's Law:** Computational power doubles roughly every two years.

⟹ *Exponential time algorithms will never be able to solve large problems.*




# Software versus Hardware

- Thus, improvements in hardware will never overcome *bad algorithm design*.
- Indeed, the current dramatic breakthroughs in computer science are based upon better (faster and higher performance) algorithmic techniques.

# Computational Complexity

- Therefore, a basic aim of computer science is to understand which problems have **good algorithms** and which problems don't.
- As a <u>first</u> step, we do this by assigning problems to complexity classes in the computational hierarchy.

*e.g.* **P** The set of decision problems that can always be solved in polynomial time by a (deterministic) computer.

**NP** The set of decision problems that can always be solved in polynomial time by a non-deterministic computer.

The set of decision problems that can be solved in exponential time by Exhaustive Search.

- The hardest problems in these groups form an equivalence class called **complete** problems.

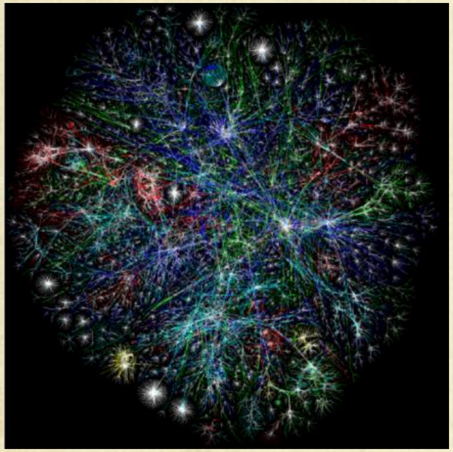
# The $P \neq NP$ Conjecture

- This conjecture states that <u>computation is harder than verification</u>.

**Informally 1**: If $P = NP$ then the existence of a bad algorithm (i.e. exhaustive search) implies the existence of a good algorithm.

**Informally 2**: If $P = NP$ then having a wizard who can magically find you an optimal solution does not help you in your search!

- Traditionally, computer scientists have studied computation in the context of an *optimizer* designing fast code in problem solving.
- However, recently there has been an strong interest in dealing with situations where there *multiple decision-makers*.






- Why should Economists or Game Theorists or Bayesian Persuaders care about computation?
  - ~~Because it explains why every example in economics textbooks and academic papers have only two agents and at most two time periods!~~
  - Because it applies beyond computers to every decision-maker and decision-making process and to all mechanisms and markets.
- For example, we have no **good algorithms** for
  - *Nash equilibria in Bimatrix Games.*
  - *Combinatorial Auctions.*
  - *Market Equilibria.*
  - *Fair Division, etc.*

# Comments and Questions

- The main result shows, for any monotone set function $f$, a computational equivalence between optimizing the private signal and the maximizating $f$ plus an additive function.
  - *The proof is cleverly pieced together in via the techniques of reduction, LP duality and separation oracles.*
- Bayesian Persuasion relates very closely to Correlated Equilibria.
  - *Can you explain why Bayesian persuasion is a more general problem?*
  - *The techniques used in the paper also relate closely to approaches taken to find welfare maximizing correlated equilibria.*
    - *How do your methods differ?*
    - *Can your methods be used to obtain stronger results for correlated equilibria or other problems?*