The Limits of Symmetric Computation

Anuj Dawar

Department of Computer Science and Technology, University of Cambridge

Archimedeans, 22 November 2019

P vs. NP

The P vs. NP problem is the *most famous* problem in theoretical computer science.

It is one of six remaining *Clay Millenium Prize* problems.

Research motivated by this question has spawned a vast field of work in *Complexity Theory*.

Algorithmic Problems

P the class of problems solvable *efficiently*.

the number of steps required by an algorithm to solve it grows polynomially in the instance size.

NP the class of problems for which a solution can be *checked efficiently*.

there is an algorithm, given an instance and a candidate solution can check it using a number of steps that grows polynomially in the the instance size.

Example

Consider a system of linear equations:

$$\begin{array}{rcl} a_{11}x_1 + \cdots a_{1n}x_n & = & b_1 \\ a_{21}x_1 + \cdots a_{2n}x_n & = & b_2 \\ \vdots & & \vdots \\ a_{m1}x_1 + \cdots a_{mn}x_n & = & b_m \end{array}$$

The *instance* is the matrix A and the vector b, and we wish to know if there is an x such that $Ax = b$.

What do the variables range over?

Given a matrix A and vector b over the rationals $\mathbb{Q}$, does there exist a rational vector x with $Ax = b$?

The problem is in P using the Gaussian elimination algorithm.
This requires proving that the bit complexity of the solution is bounded by a polynomial in that of the instance.

The same argument works for A, b and x over a *finite field* K.

Given a matrix A and vector b over the integers $\mathbb{Z}$, does there exist an integer vector x with $Ax = b$?

Now Gaussian elimination does not work.
Nonetheless the problem is in P by other algorithms.

The same argument works for A, b and x over a *finite ring* R.

The Natural Numbers

Given a matrix A and vector b over $\mathbb{N}$, does there exist a non-negative integer vector x with $Ax = b$?

> *The problem is in NP because we can bound the value of a solution by an exponential function of the instance.*
> *We know of no polynomial-time algorithm for the problem.*

Indeed, the problem is *NP-complete* meaning that a *polynomial-time* algorithm would imply P = NP.

The problem is already NP-complete even if we are looking for solutions in $\{0, 1\}$.

NP-completeness

A problem in NP has an *exponential size* search space of possible solutions.

> *E.g., the 2^n possible $\{0,1\}$-values of the n unknowns in the vector x.*

Sometimes the *algebraic structure* of the problem means we can converge quickly to a solution, and so the problem is in P.

> *E.g, systems $Ax = b$ where addition and multiplication are taken modulo 2.*

Sometimes the lack of structure means we can code *any* problem in NP in the solution space of an instance, and the problem is NP-complete.

> *E.g., any set of the 2^n $\{0,1\}$-vectors can occur as the solution set of $Ax = b$ over the integers.*

Graph Problems

Among the most commonly studied algorithmic problems are problems on *graphs*.

Some problems in P:

> *Eulerian Graphs*: *Given a graph $G = (V, E)$, is there a walk starting at a vertex v, returning to v and passing through every edge exactly once.*
>
> *Perfect Matching*: *Given a graph $G = (V, E)$, is there a subset $M \subseteq E$ such that each $v \in V$ is incident on exactly one edge in M.*

Graph Problems

Some NP-complete graph problems:

Hamiltonicity: *Given a graph $G = (V, E)$, is there a cycle starting at a vertex v, returning to v and passing through every vertex exactly once.*

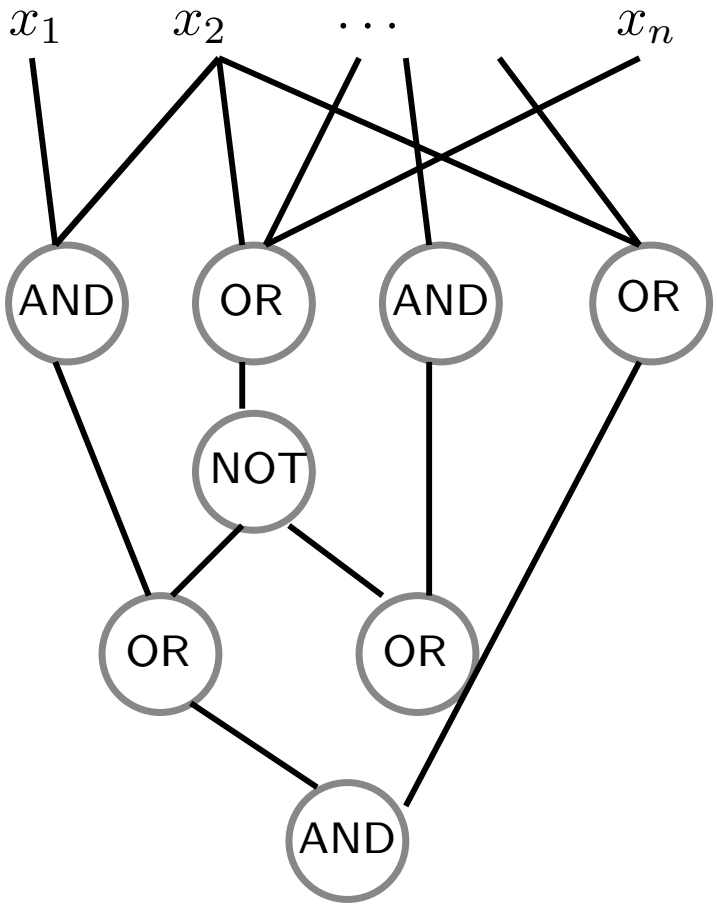
3-colourability: *Given a graph $G = (V, E)$, is there a function $\chi : V \to \{1, 2, 3\}$ such that $(u, v) \in E \Rightarrow \chi(u) \neq \chi(v)$*

Circuit Models

How could we prove the *impossibility* of an algorithm?

Any *polynomial-time* algorithm gives, for each *input size* a *circuit*:

Circuits are just the *unfoldings* of the behaviour of an algorithm on inputs of a fixed size n into simple actions such as Boolean *AND*, *OR* and *NOT* operations.



P/poly

P/poly is the class of problems for which, for each value of n, there is a circuit of *size polynomial in* n which correctly decides the problem.

It is conjectured that NP $\not\subseteq$ P/poly.

This means that it is not possible to solve an NP-complete problem even if we allow

- an arbitrary amount of computation based on the *size* of the input;
- followed by a polynomial amount of computation given the actual input.

Monotone Problems

Some graph problems are naturally *monotone*.

If $G = (V, E)$ and $H = (V, E')$ are graphs with $E \subseteq E'$ and G contains a *Hamiltonian cycle*, then so does H.

3-colourability is not monotone but its *complement* is:

> *If $G = (V, E)$ is not 3-colourable, then neither is $H = (V, E')$ when $E \subseteq E'$.*

In principle, these can be decided by families of *monotone* circuits, i.e. using only *AND* and *OR* gates.

Circuit Lower Bounds

For some *monotone* problems in NP, we can prove that no *polynomial-size* family of *monotone* circuits suffices to decide the problem.

- No *polynomial-size* family of *monotone* circuits decides *clique*.
- No *polynomial-size* family of *monotone* circuits decides *perfect matching*.

(Razborov 1985).

Lower bounds have also been established by restricting the *depth* of circuits.

- No *constant-depth* (unbounded fan-in), *polynomial-size* family of circuits decides *parity*. **(Furst, Saxe, Sipser 1983)**.
- No *constant-depth*, $O(n^{\frac{k}{4}})$-*size* family of circuits decides *k-clique*.

(Rossman 2008).

Circuits for Graph Problems

We want to study families of circuits that decide properties of *graphs* (or other relational structures—for simplicity of presentation we restrict ourselves to graphs).

We have a family of Boolean circuits $(C_n)_{n\in\omega}$ where there are n^2 inputs labelled $(i,j) : i,j \in [n]$, corresponding to the *potential edges*.
Each input takes value 0 or 1;

Graph properties in P are given by such families where:

- the size of C_n is bounded by a polynomial $p(n)$; and
- the family is uniform, so the function $n \mapsto C_n$ is in P.

Invariant Circuits

C_n is *invariant* if, for every input graph, the output is unchanged under a permutation of the inputs induced by a permutation of $[n]$.

That is, given any input $G : [n]^2 \to \{0, 1\}$, and a permutation $\pi \in S_n$,

C_n *accepts* G *if, and only if,* C_n *accepts the input* πG *given*

$$(\pi G)(i, j) = G(\pi(i), \pi(j)).$$

Note: this is not the same as requiring that the result is invariant under *all* permutations of the input. That would only allow us to define functions of the *number* of 1s in the input. The functions we define include all *isomorphism-invariant* graph properties such as *Eulerian graphs*, *perfect matching*, *Hamiltonicity*, *3-colourability*.

Symmetric Circuits

Say C_n is *symmetric* if any permutation of $[n]$ applied to its inputs can be extended to an automorphism of C_n.

> *i.e., for each $\pi \in S_n$, there is an automorphism of C_n that takes input (i, j) to $(\pi i, \pi j)$.*

Any symmetric circuit is invariant, but *not* conversely.

FPC

FPC is a class of *decision problems* definable in *fixed-point logic with counting*.

> *The decision problems are (isomorphism-closed) classes (or properties) of finite structures (such as graphs, Boolean formulas, systems of equations).*

A graph property is in FPC *if, and only if,* it is decided by a P-uniform family of *symmetric* circuits using *AND*, *OR*, *NOT* and *MAJ* gates.

Excluding *MAJ* gates gives us something *strictly weaker*.

Symmetric Computation

Say a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *symmetric* if it is invariant under *all* permutations of its inputs.

A graph property is in FPC *if, and only if,* it is decided by a P-uniform family of *symmetric* circuits using *symmetric gates*.

FPC gives a natural notion of *polynomial-time, symmetric* computation.

Impossibility Results

Some NP-complete problems are *provably* not in FPC, including:

- *Sat*
- *Hamiltonicity*
- *3-colouraiblity*

For some NP-complete problems, inclusion in FPC is an open problem, equivalent to P = NP.

The Power of FPC

Most *"obviously"* polynomial-time algorithms can be expressed in FPC.

Many non-trivial polynomial-time algorithms can be expressed in FPC:

- FPC captures all of P over any *proper minor-closed class of graphs* **(Grohe 2017)**
- In FPC we can express the existence of a *Eulerian cycle* or a *perfect matching*.
- Solving systems of equations over the *rationals* or the *integers*.
- Optimization algorithms based on *linear programming* and *semidefinite programming*.

But This Doesn't Settle the Question

But some cannot be expressed:

- There are polynomial-time decidable properties of graphs that are not definable in FPC. **(Cai, Fürer, Immerman, 1992)**
- *XOR-Sat*, or more generally, solvability of a system of linear equations over a finite field cannot be expressed in FPC.

In particular, this means that the Gaussian elimination algorithm cannot be made symmetric without a super-polynomial blow-up.

Proving Impossibility

To show that some property P of graphs cannot be determined by a family of *polynomial-size* symmetric circuits we use:

A support theorem: *This characterizes the groups of symmetries occuring in a symmetric circuit*

Approximations of isomorphism: *Certain equivalence relations* $G \equiv H$ *on graphs weaker than isomorphism.*

Non-invariance: *Showing that the property* P *is not invariant under the equivalence relation* $\equiv$.

Stabilizers

For a symmetric circuit C taking n-vertex graphs as input, we can assume *w.l.o.g.* that the automorphism group is the symmetric group S_n acting in the natural way.

For a gate g in C, $\mathrm{Stab}(g)$ denotes the *stabilizer group of* g, i.e.,

$$\mathrm{Stab}(g) = \{\pi \in S_n \mid \pi(g) = g\}.$$

By the *orbit-stabilizer* theorem, the size of the *orbit* of any gate g in C is $\frac{n!}{|\mathrm{Stab}(g)|}$.

So, an upper bound on $\mathrm{Stab}(g)$ gives us a lower bound on the orbit of g.

Conversely, knowing that the orbit of g is at most polynomial in n gives us bounds on $\mathrm{Stab}(g)$.

Supports

In a symmetric circuit C taking n-vertex graphs as input, say a set $X \subseteq [n]$ is a *support* of a gate g if

> *every* $\pi \in S_n$ *which fixes* X *pointwise is in* $\mathrm{Stab}(g)$.

For example, the *output gate* has empty support.

An input gate corresponding to (i, j) has support $\{i, j\}$.

We are able to show that, if C has size $O(n^k)$, then *every* gate in C has a support X with $|X| = O(k)$.

Approximations of graph isomorphism

The *graph isomorphism*—given two graphs is there a bijection between the two sets of vertices that preserves the edges—is not known to be in P.

The k-*dimensional Weisfeiler-Leman* equivalence relation is an *overapproximation* of the isomorphism relation.

If G, H are n-vertex graphs and $k < n$, we have:

$$G \cong H \quad \Leftrightarrow \quad G \equiv^n H \quad \Rightarrow \quad G \equiv^{k+1} H \quad \Rightarrow \quad G \equiv^k H.$$

$G \equiv^k H$ is decidable in time $n^{O(k)}$.

It has many equivalent characterisations arising from *combinatorics*; *logic*; *algebra*; *linear optimization*.

Pebble Games

Pebble Games are two-player games that are used to define *equivalence relations* on structures characterising forms of *indistinguishability*.

We are particularly interested in the *bijection game* **(Hella 96)**.

The game is played by two players *Spoiler* and *Duplicator* on structures (*e.g.* graphs) $\mathbb{A}$ and $\mathbb{B}$ with pebbles $a_1, \ldots, a_k$ on $\mathbb{A}$ and $b_1, \ldots, b_k$ on $\mathbb{B}$.

- *Spoiler* chooses a pair of pebbles a_i and b_i;
- *Duplicator* chooses a bijection $h : V^{\mathbb{A}} \to V^{\mathbb{B}}$ such that for pebbles a_j and $b_j (j \neq i)$, $h(a_j) = b_j$;
- *Spoiler* chooses $a \in V^{\mathbb{A}}$ and places a_i on a and b_i on $h(a)$.

Duplicator loses if the partial map $a_i \mapsto b_i$ is not a partial isomorphism. It turns out that $G \equiv^k H$ *if, and only if, Duplicator* has a strategy to play forever.

Circuits and Pebble Games

We can use *bijection games* and the *support theorem* to establish lower bounds for symmetric circuits.

The key is the following connection.

> *If C is a symmetric circuit on n-vertex graphs such that every gate of C has a support of size at most k, and G and H are graphs such that $G \equiv^k H$ then:*
>
> *C accepts G if, and only if, C accepts H.*

This can be proved by showing that if C distinguishes G from H, then it provides a *winning strategy* for *Spoiler* in the k-pebble bijection game.

Invariance

To show that some property P of graphs cannot be determined by *symmetric, polynomial-size* circuits, it suffices to show that it is *not invariant* under $\equiv^k$ for any fixed k.

That is, for each k, we can find a pair of graphs G and H, which differ on P, but $G \equiv^k H$.

We can do this for many *decision problems*: *Hamitonicity, 3-colourability,*

And for some *numerical parameters*: *minimum vertex cover*, *maximum clique*, *number of perfect matchings*.

Sometimes, even up to a *multiplicative factor*.

This means that none of these properties can be computed (or, in some cases even approximated) by polynomial-time symmetric algorithms.

Systems of equations

Given a matrix A and vector b over the field K, does there exist a vector x with $Ax = b$?

This *can* be solved by a polynomial-time symmetric algorithm when $K = \mathbb{Q}$, but *provably not* for any finite field K.

The *ellipsoid method* for solving linear inequalities can be implemented symmetrically, but *Gaussian elimination* cannot.

Limits of Symmetric Computation

FPC defines a natural notion of *symmetric polynomial-time computation*.

It is remarkably powerful and able to express many *non-trivial* polynomial-time algorithms.

These include some of the strongest algorithmic techniques for approximating NP-hard optimization problems.

Since we are able to show for some NP-hard optimization problems that *no* algorithm expressible in FPC can solve them exactly, we establish limitations on commonly used approximation techniques.

For many, we are also able to prove that no algorithm in FPC can solve them even *approximately*.

A Rich Theory of Symmetry in Computation

A number of *distinct strands* of research are *converging* on a study of *symmetry in computation*.

Besides those mentioned here, there is work on the complexity of *constraint satisfaction problems*; of symmetry in *combinatorial optimization*; of *semi-structured data* and *abstract syntax*.

The research builds on a combination of *algebraic*, *logical* and *combinatorial* methods.

An exciting, emerging field in theoretical computer science, dealing with both *abstraction* and *complexity*.