The Limits of Symmetric Computation

Anuj Dawar

Department of Computer Science and Technology, University of Cambridge

Zhejiang University, 21 June 2021

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 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 $Ax = b$.

The *size* of the instance is the *number of bits* required to write down all the numbers in A and 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 the $\mathbb{Z}$, 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 $\mathsf{P} = \mathsf{NP}$.

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.*

Boolean Satisfiability

The classic NP-complete problem is the *satisfiability* of Boolean formulas in conjunctive normal form (*SAT* for short).

> *Each formula is the AND of clauses, where each clause is the OR of a number of literals.*

On the other hand, *XOR-SAT* is solvable in polynomial time.

> *Each formula is the AND of clauses, where each clause is the XOR of a number of literals.*

This is essentially the same as solving systems of equations over the 2-element field.

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.

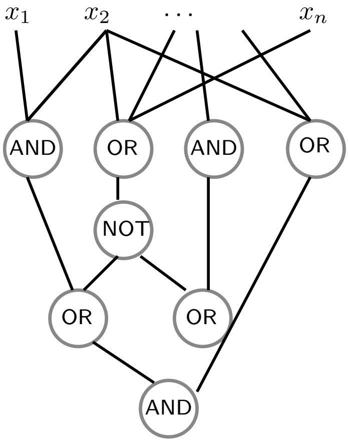
3-colourability: Given a graph $G = (V, E)$, is there a function $\chi : V \rightarrow \{1, 2, 3\}$ such that $(u, v) \in E \Rightarrow \chi(u) \neq \chi(v)$

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 $\mathsf{NP} \not\subseteq \mathsf{P}/\mathrm{poly}$.

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.

Lower Bounds

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 $\mathsf{P} = \mathsf{NP}$.

Upper Bounds

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*.

Lower Bounds

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.

Fixed-Point Logic with Counting

FPC is a logic formulated to add *inductive definitions* and the ability to *count* to first-order logic (FO).

If $\varphi(x)$ is a formula with free variable x, then $\#x\varphi$ is a term denoting the number of elements satisfying φ.

Formulae of FPC:

- all atomic formulae as in FO;
- $\tau_1 < \tau_2$; $\tau_1 = \tau_2$ where τ_i is a term of numeric sort;
- $\exists x\, \varphi$; $\exists \nu\, \varphi$; where ν is a variable ranging over numbers up to the size of the domain;
- $[\mathbf{lfp}_{X,\mathbf{x},\nu}\varphi](\mathbf{t})$; and
- $\varphi \land \psi$; $\neg\varphi$.

Counting Quantifiers

C^k is the logic obtained from *first-order logic* by allowing:

- *counting quantifiers*: $\exists^i x\, \varphi$; and
- only the variables $x_1, \ldots . x_k$.

Every formula of C^k is equivalent to a formula of first-order logic, albeit one with more variables.

For every sentence φ of FPC, there is a k such that φ is equivalent to a *theory* of C^k.

Indeed, for any fixed n, there is a formula of C^k equivalent to φ on structures with at most n elements.

Weisfeiler-Leman

For a pair of graphs G and H, we write $G \equiv^k H$ to denote that they are not distinguished by any sentence of C^k.

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

It has many equivalent characterisations arising from

- *combinatorics* **(Babai)**
- *logic* **(Immerman-Lander)**
- *algebra* **(Weisfeiler; Holm)**
- *linear optimization* **(Atserias-Maneva; Malkin)**

Counting Width

For any class of structures $\mathcal{C}$, we define its *counting width* $\nu_{\mathcal{C}} : \mathbb{N} \to \mathbb{N}$ so that

> $\nu_{\mathcal{C}}(n)$ *is the least* k *such that* $\mathcal{C}$ *restricted to structures with at most* n *elements is closed under* $\equiv^k$.

More generally, let μ be a *numeric graph parameter*.

That is, it assigns a *numeric value* $\mu(G)$ to any graph G.

The *counting width* of μ is the function $\nu_\mu : \mathbb{N} \to \mathbb{N}$ such that

> $\nu_\mu(n)$ *is the least* k *such that for* n*-vertex graphs* G *and* H*,* $G \equiv^k H$ *implies* $\mu(G) = \mu(H)$.

Counting Width

Every class definable in FPC has counting width bounded by a *constant*.

Also, any *numeric parameter* definable in FPC has counting width bounded by a constant.

To say a class has *constant counting width* is the same as saying it is *axiomatizable* in C^k for some constant k.

Many natural problems can be shown to have *unbounded* counting width. They are, *hence* not definable in FPC.

3SAT, *XOR-Sat*, *Hamiltonicity*, *3-Colourability* all have counting width $\Omega(n)$.

Linear Programming

Linear Programming is an important algorithmic tool for solving a large variety of optimization problems.

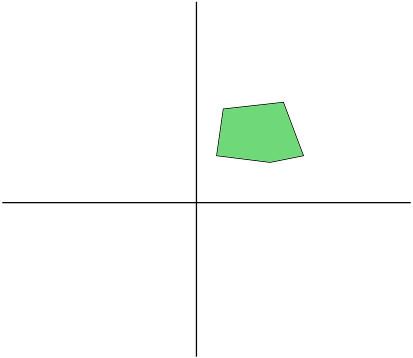
It was shown by **(Khachiyan 1980)** that linear programming problems can be solved in polynomial time.
We have a set C of *constraints* over a set V of *variables*.
Each $c \in C$ consists of $a_c \in \mathbb{Q}^V$ and $b_c \in \mathbb{Q}$.

Feasibility Problem: Given a linear programming instance, determine if there is an $x \in \mathbb{Q}^V$ such that:

$$a_c^T x \leq b_c \quad \text{for all } c \in C$$

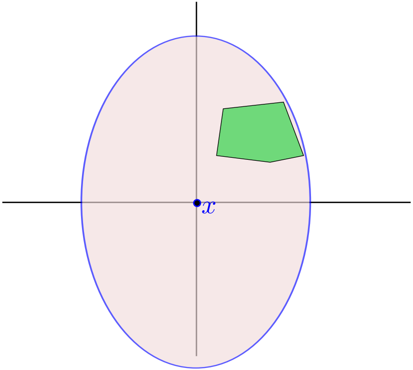
This, and the corresponding *optimization problem* are expressible in FPC.

Ellipsoid Method



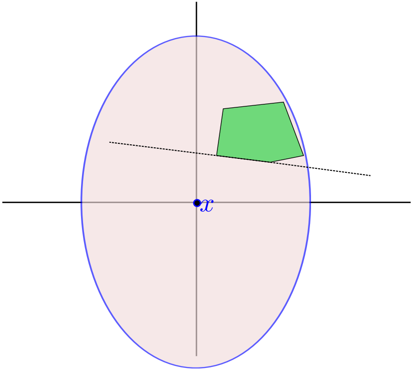
The set of constraints determines a *polytope*

Ellipsoid Method



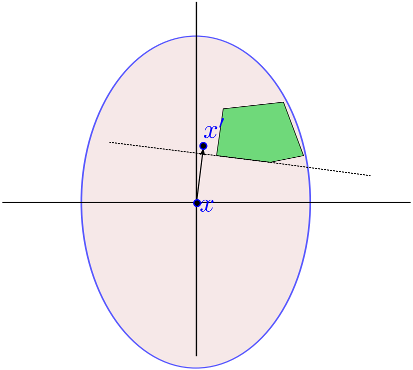
Start at the origin and calculate an *ellipsoid* enclosing it.

Ellipsoid Method



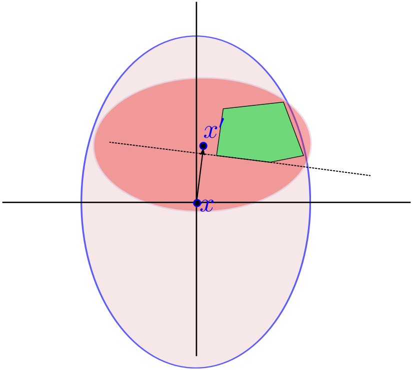
If the centre is not in the polytope, choose a constraint it *violates*.

Ellipsoid Method



Calculate a new *centre*.

Ellipsoid Method



And a new ellipsoid around the centre of at most *half* the volume.

Ellipsoid Method in FPC

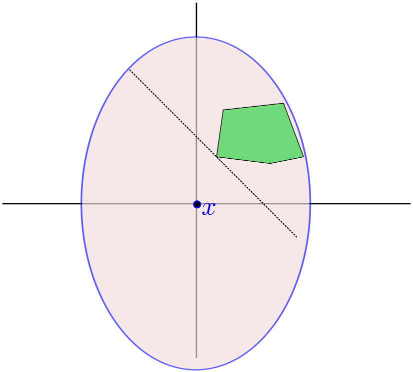
We can encode all the calculations involved in FPC.

This relies on expressing algebraic manipulations of *unordered* matrices.

What is not obvious is how to *choose* the violated constraint on which to project.

However, the ellipsoid method works as long as we can find, at each step, some *separating hyperplane*.

Ellipsoid Method in FPC



Ellipsoid Method in FPC

We can encode all the calculations involved in FPC.

This relies on expressing algebraic manilpulations of *unordered* matrices.

What is not obvious is how to *choose* the violated constraint on which to project.

However, the ellipsoid method works as long as we can find, at each step, some *separating hyperplane*.

So, we can take:

$$(\sum_{c \in S} a_c)^T x \leq \sum_{c \in S} b_c$$

where S is the *set* of all violated constraints.

Linear Programs for Hard problems

In the 1980s there was a great deal of excitement at the discovery that *linear programming* could be done in *polynomial time*.

This raised the possibility that linear programming techniques could be used to *efficiently* solve hard problems.

Many proposals were put forth for encoding *hard* problems (such as the *Travelling Salesman Problem*) (TSP) as linear programs.

(Yannakakis 1991) proved that *any* encoding of TSP as a linear program, satisfying natural *symmetry* conditions, must have *exponential size*.

Travelling Salesman Problem

Given a set of V of n vertices and a distance matrix $C = \mathbb{R}^{V \times V}$, find

$$\min_{\pi \in [n] \overset{\text{bij}}{\to} V} \sum_{i \in [n]} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)}$$

To formulate this as a *linear optimization* problem, introduce a set of variables:

$$X = \{x_{ij} \mid i, j \in V\}.$$

So, a graph is a *function* $G : X \to \{0, 1\}$.
Let $P \subseteq \{0, 1\}^X$ be the collection of simple cycles of length n.

TSP polytope

Let $\mathsf{conv}(P) \subseteq \mathbb{R}^X$ be the *convex hull* of P.
That is, the set of $\vec{y} \in \mathbb{R}^X$ such that

$$\vec{y} = \sum_{\vec{x} \in P} \lambda_{\vec{x}} \vec{x} \quad \text{with } \lambda_{\vec{x}} \geq 0 \text{ and } \sum_{\vec{x} \in P} \lambda_{\vec{x}} = 1.$$

TSP: $\min \sum_{i,j \in V} c_{ij} x_{ij}$ over $\vec{x} \in P$.
This is equivalent to minimizing $\sum_{i,j \in V} c_{ij} x_{ij}$ over $\mathsf{conv}(P)$.

We call $\mathsf{conv}(P)$ the *TSP polytope*.

$\mathsf{conv}(P)$ has *exponentially many facets*.

Extended Formulations

Could $\mathsf{conv}(P)$ be obtained as the *projection* of a polytope with a small number of facets?

Is there a *small* $Q \subseteq \mathbb{R}^X \times \mathbb{R}^Y$ such that

$$\{\vec{x} \mid \exists \vec{y}(\vec{x}, \vec{y}) \in Q\} = \mathsf{conv}(P)?$$

If a description of such a Q could be obtained in *polynomial time* in n, then $\mathsf{P} = \mathsf{NP}$.

If such a Q of *polynomial size* exists, then $\mathsf{NP} \subseteq \mathsf{P}/\mathrm{poly}$.

Also note that by adding inequalities $x \leq G(x)$ for a graph $G : X \to \{0, 1\}$, we obtain a polytope $Q_G \subseteq \mathbb{R}^X \times \mathbb{R}^Y$ which is *non-empty* if, and only if, G contains a Hamiltonian cycle.

Yannakakis

Say $Q \subseteq \mathbb{R}^X \times \mathbb{R}^Y$ is *symmetric* if for every $\pi \in S_V$, there is a $\sigma \in S_Y$ such that

$$Q^{(\pi,\sigma)} = Q$$

Here, we extend the action of π to $V \times V$, and hence to $\mathbb{R}^X$. similarly σ to $\mathbb{R}^Y$.

Theorem (Yannakakis)
Any symmetric $Q \subseteq \mathbb{R}^X \times \mathbb{R}^Y$ whose projection on $\mathbb{R}^X$ is $\mathsf{conv}(P)$ has *exponentially* many facets.

This is derived from a similar lower bound for the *matching polytope*.

Symmetric Linear Programs

Fix $X = \{x_{ij} \mid i, j \in [n]\}$ for a fixed n.
Consider a class $\mathcal{C}$ of graphs.
We identify a graph on n vertices with a function $G : X \to \{0, 1\}$.

We say that a polytope $Q \subseteq \mathbb{R}^X \times \mathbb{R}^Y$ *recognizes* $\mathcal{C}$ if its projection on $\mathbb{R}^X$ includes $\mathcal{C}|_n$ and excludes its complement.

Say $Q \subseteq \mathbb{R}^X \times \mathbb{R}^Y$ is *symmetric* if for every $\pi \in S_V$, there is a $\sigma \in S_Y$ such that

$$Q^{(\pi,\sigma)} = Q$$

Here, we extend the action of π to $V \times V$, and hence to $\mathbb{R}^X$.

Symmetric Linear Programs

Theorem (Atserias, D., Ochremiak '19)
If a family of symmetric polytopes of size $s = O(2^{n^{1-\epsilon}}), \epsilon > 0$ recognizes $\mathcal{C}$, then $\mathcal{C}$ has *counting width* at most $O(\frac{\log s}{\log n})$.

In particular, classes of counting width $\Omega(n)$ are not recognized by any *subexponential* size symmetric linear programs.

We get an *exponential* lower bound on the size of any symmetric extended formulation of *Hamiltonicity*

In contrast, the class of graphs with a *perfect matching* does have *bounded counting width*. Indeed, it is definable in FPC.

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.

Arithmetic Circuits

Arithmetic Circuits over a field K have:

- Inputs labelled by a *variable* $x \in X$, or constant $c \in K$.
- Internal gates labelled by $+$ or $\times$.

Each circuit computes a *polynomial* in $K[X]$.

Valiant's conjecture VP $\neq$ VNP is that there are no *polynomial-size arithmetic circuits* for computing the *permanent*

$$\sum_{\sigma \in S_n} \prod_{i \in [n]} x_{i\sigma(i)}$$

Note: there are such circuits for the *determinant*:
$\sum_{\sigma \in S_n} (-1)^{\mathsf{sgn}(\sigma)} \prod_{i \in [n]} x_{i\sigma(i)}$

Symmetric Arithmetic Circuits

Both the *determinant* and the *permanent* are defined over a set of variables $x_{ij} \,:\, i, j \in \{1, \ldots, n\}$.

Both are invariant under *permutations* of the variables induced by the action of S_n on $\{1, \ldots, n\}$.

Are they computed by *symmetric, polynomial-size, arithmetic* circuits?

We are able to prove that the determinant *is* and the permanent *provably is not*. **(D. Wilsenach 2020)**

This is proved by showing that the *number of perfect matchings in a bipartite graph* on n vertices has counting width $\Omega(n)$.

A Rich Theory of Symmetry in Computation

A number of *distinct strands* of research *converge* 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 heavily on mathematical tools for the study of symmetry: *group theory*.

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