

When and why do efficient algorithms exist (for constraint satisfaction and beyond)?

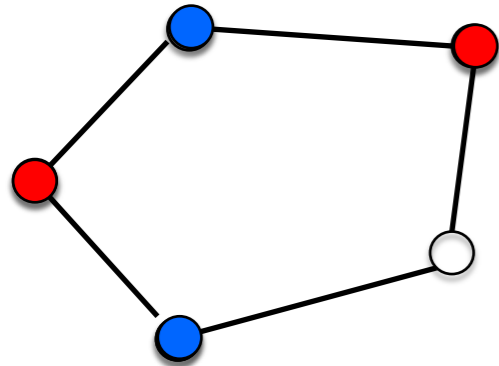
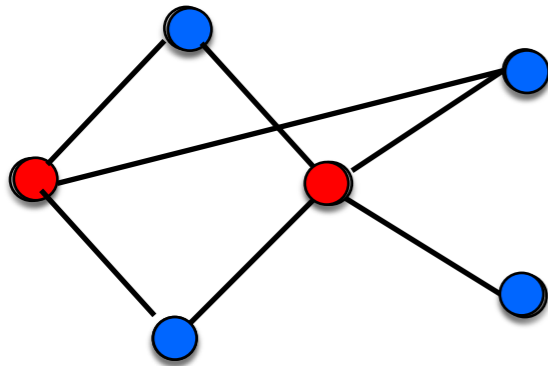
Venkatesan Guruswami



-- University of Warsaw --
June 30, 2025

An easy task: 2-coloring graphs

Can one color vertices of a graph **Red** / **Blue** so that vertices connected by an edge don't get the same color



Simple propagation algorithm:

- Color some node **Red**,
- Colors its neighbors **Blue**,
- Their neighbors **Red**, and so on

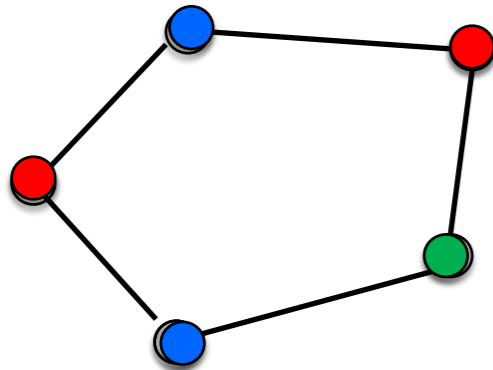
When done, check no inconsistency / miscolored edge.

2-coloring is an **easy** task.

$O(n)$ time complexity, $n =$ size of graph

3-coloring graphs

Can one color vertices of a graph **Red** / **Blue** / **Green** so that vertices connected by an edge don't get the same color



After coloring a vertex, color of its neighbor not fixed, and has 2 choices

Combinatorial explosion of possibilities to cover all bases.

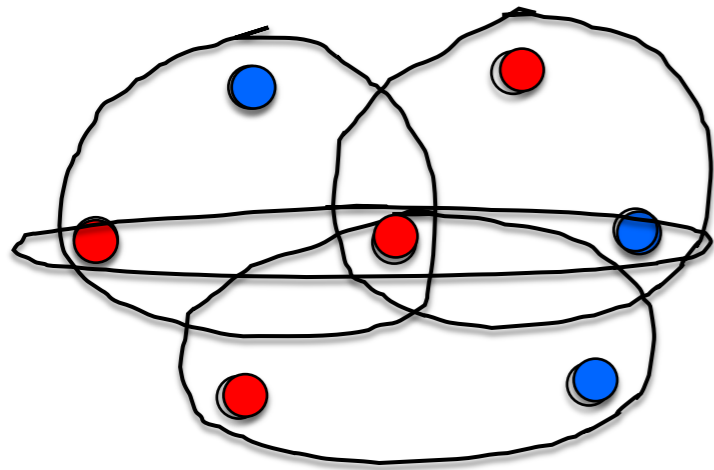
All known methods for 3-coloring take *exponential time*

- Record: c^n for $c \approx 1.329$

3-coloring graphs is an **NP-hard** problem

A hard form of 2-coloring

2-coloring a 3-uniform hypergraph



Assign vertices **Red** / **Blue** so that *no triple is monochromatic*.

Also a classic NP-hard problem.

Complexity dichotomy

- 2-coloring easy, solvable in polynomial (i.e., n^c) time
- 3-coloring (or 2-coloring 3-hypergraphs) NP-hard, c^n runtime

Polynomial (i.e., n^c) vs. exponential (i.e., c^n) : a big chasm in complexity.

- *Dichotomous* behavior

Today's story:

Dichotomy for a natural class of problems;
an explanation of what governs it;
and possible roads ahead

Can we “explain” complexity?

Why are some problems easy and some hard?

What underlying mathematical structure (or lack thereof) in a computational problem enables an efficient algorithm (or dictates its hardness)?

Given the vast scope of problems & clever algorithms, shouldn't expect a universal answer ...

But major success for **CSP** and some of its variants.

Structure governing complexity

Philosophy: The *structure and symmetries in the solution space* of a problem can shed light on its tractability.

Polymorphism, an operation under which the space of solutions is closed.

Let's ease in with a familiar example

Linear equations

Fix a familiar ring R .

Eg. $\mathbb{Q}, \mathbb{Z}, \mathbb{F}_p$,

System of linear equations

$$Ax = b, \text{ with } A \in R^{m \times n}, b \in R^m$$

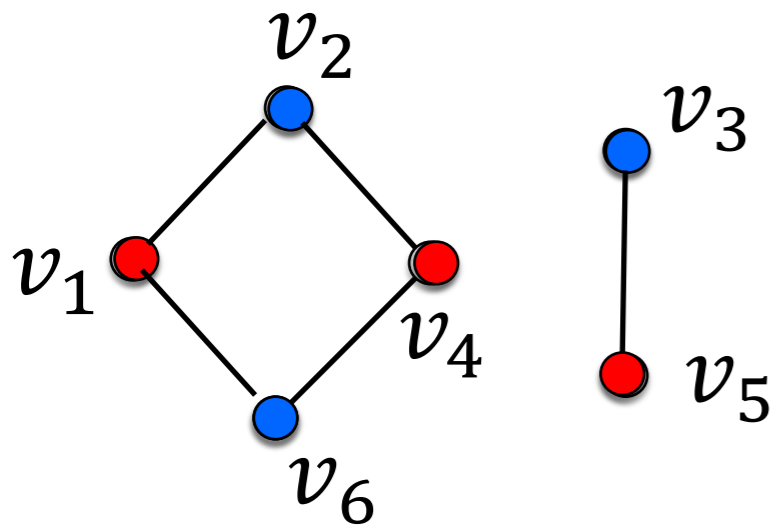
Closure property: If x_1, x_2, x_3 are three solutions to the system, then so is $x_1 - x_2 + x_3$

$f : R^3 \rightarrow R$ defined by $f(a, b, c) = a - b + c$ is a *polymorphism* for the linear equations problem

Efficiently solvable by Gaussian Elimination/ other algos.

(Discrete) polymorphisms: 2-coloring

2-coloring instance:



	v_1	v_2	v_3	v_4	v_5	v_6
	1	0	0	1	1	0
	0	1	1	0	0	1
	1	0	1	1	0	0

Maj_3 : 1 0 1 1 0 0

Odd_3 : 0 1 0 0 1 1

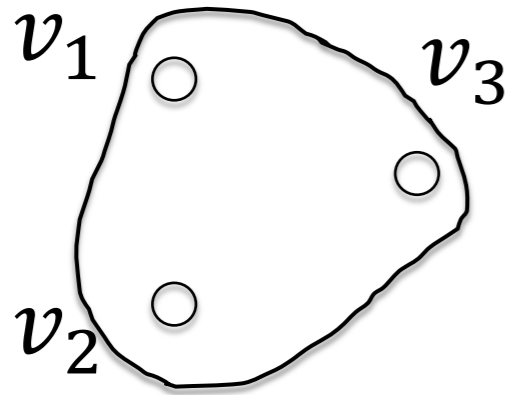
Closure property: For all valid colorings c_1, c_2, c_3 ,
Majority(c_1, c_2, c_3) (coordinate-wise majority) is also a valid coloring

$Maj_3: \{0,1\}^3 \rightarrow \{0,1\}$ is a polymorphism for 2-coloring.

So is $Odd_3: \{0,1\}^3 \rightarrow \{0,1\}$; $Odd_3(a, b, c) = a + b + c \pmod{2}$

Efficiently solvable by various algorithms

Polymorphisms: 2-coloring 3-hypergraphs



Not-all-equal(v_1, v_2, v_3)

	v_1	v_2	v_3	
	1	0	0	
	0	1	0	
	0	0	1	
Maj_3 :	0	0	0	(oops!)

There are **no** interesting* polymorphisms in this case

Same holds for graph 3-coloring

And these problems are NP-hard

* : can always copy a coloring, or flip it

Polymorphisms: local operations to combine *multiple* solutions and produce a new solution

For the rich class of **constraint satisfaction problems** (CSP), a deep theorem asserts:

Polytime algorithms exist* if and only if there are “interesting” polymorphisms

Precise link

mathematical structure \leftrightarrow *computational complexity!*

(* under $P \neq NP$)

Constraint Satisfaction Problem (CSP)

- Finite domain D (e.g., $D = \{0,1\}$)
- Predicate $P : D^k \rightarrow \{0,1\}$
(in general, finite set $\Gamma = \{P_1, P_2, \dots\}$ of allowed predicates, called *template*)

Instance of **CSP(P)**:

- Variables $V = \{x_1, x_2, \dots, x_n\}$
- Collection of constraints:
$$P(x_1, x_3) \wedge P(x_2, x_4) \wedge P(x_1, x_{17}) \wedge \dots$$

Eg. $P(x, y) = 1$
when $x \neq y$

Coloring a graph
with $|D|$ colors

Question: Is there an assignment $\sigma: V \rightarrow D$ that satisfies all constraints? If so, find one.

Template Γ : a finite set of predicates over domain D

$\text{CSP}(\Gamma)$: constraint satisfaction problems where constraints are allowed to be drawn from Γ

Input: A formula

$$P_1(x_{i_{1,1}}, x_{i_{1,2}}, \dots, x_{i_{1,a_1}}) \wedge \dots \wedge P_m(x_{i_{m,1}}, x_{i_{m,2}}, \dots, x_{i_{m,a_m}})$$

where $P_1, P_2, \dots, P_m \in \Gamma$, over variables x_1, x_2, \dots, x_n .

Decide: Is the formula satisfiable?

- Graph 2-coloring: $D = \{0,1\}; \Gamma = \{ (x \neq y) \}$
- 3-hypergraph 2-coloring: $D = \{0,1\}; \Gamma = \{ NAE(x, y, z) \}$
- 1-in-3-SAT: $D = \{0,1\}; \Gamma = \{ (x + y + z == 1) \}$
- Graph s-t-connectivity: $D = \{0,1\}; \Gamma = \{ 0, 1, (x \rightarrow y) \}$
- Graph- k -Coloring: $D = \{1, \dots, k\}; \Gamma = \{ (x \neq y) \}$
- Unique-Games: $\Gamma = \{ \mathbf{1}(x = \sigma(y)) \mid \sigma \in \text{Perm}(D) \}$

Polymorphism

Let $P: D^k \rightarrow \{0,1\}$ be a predicate.

A function $f: D^m \rightarrow D$ is a *polymorphism* of P if *whenever*

$(x_1^{(1)}, x_2^{(1)}, \dots, x_k^{(1)})$ satisfies P

$(x_1^{(2)}, x_2^{(2)}, \dots, x_k^{(2)})$ satisfies P

\vdots

$(x_1^{(m)}, x_2^{(m)}, \dots, x_k^{(m)})$ satisfies P

$f \Downarrow \quad f \Downarrow \quad \dots \quad f \Downarrow$

(y_1, y_2, \dots, y_k) satisfies P

Set of such f
is $\text{Pol}(P)$

Uninteresting polymorphisms:

Dictator functions

- for some $i \in [m]$,

$$f(z) = z_i$$

Combines any m solns. to P to a solution of P (i.e., a **homomorphism** $P^m \rightarrow P$)

$f: D^m \rightarrow D$ is a *polymorphism* for predicate $P: D^k \rightarrow \{0,1\}$ if

- For every $m \times k$ matrix whose m *rows* satisfy P , applying f *column-wise* yields a k -tuple satisfying P .

2-coloring:

$f = \text{Maj}_3$ is a polymorphism

$$P(x, y) = 1 \text{ when } x + y = 1$$

$$P(x, y, z) = x \oplus y \oplus z$$

(odd number of 1's)

$$f = \oplus_5$$

$$0 \ 0 \ 1$$

$$1 \ 0 \ 0$$

$$1 \ 1 \ 1$$

$$0 \ 1 \ 0$$

$$1 \ 0 \ 0$$

$$1 \ 0 \ 0 \quad f \text{ applied column-wise}$$

1-in-3-SAT:

$$P(x, y, z) = \mathbf{1}[x + y + z = 1]$$

Only dictator polymorphisms.

Goal: Classify complexity of $\text{CSP}(\Gamma)$ for various Γ

Feder-Vardi **dichotomy** conjecture (1998):

For each template Γ , $\text{CSP}(\Gamma)$ is either in P or NP-complete

Schaefer's Boolean CSP dichotomy [1978]

For Boolean domain $D = \{0,1\}$

A **Boolean CSP**(Γ) is either NP-hard, or can be expressed as one of the following six types of polytime tractable problems:

1. Set everything to 0
2. Set everything to 1
3. 2-SAT
4. Horn-SAT
5. dual-Horn-SAT
6. Linear equations over \mathbb{F}_2

Feder-Vardi dichotomy conjecture (1998):

For each Γ , $\text{CSP}(\Gamma)$ is polytime solvable or NP-complete

Algebraic form of conjecture [Bulatov-Jeavons-Krokhin'05]:

$\text{CSP}(\Gamma) \in \text{P}$ if Γ admits a **non-trivial polymorphism**;
otherwise NP-complete.

Tractability governed by polymorphisms

Schaefer's theorem, polymorphic version

For Boolean domain $D = \{0,1\}$

A **Boolean CSP**(Γ) is either NP-hard, or can be expressed as one of the following six types of polytime tractable problems:

1. Set everything to 0 (**0**)
2. Set everything to 1 (**1**)
3. 2-SAT (***Maj*₃**)
4. Horn-SAT (***AND*₂**)
5. dual-Horn-SAT (***OR*₂**)
6. Linear equations over \mathbb{F}_2 (**\oplus_3**)

Algebraic CSP-dichotomy conjecture:

CSP(Γ) polytime solvable if Γ admits a **non-trivial polymorphism**; otherwise it is NP-complete.

[Zhuk'17; Bulatov'17]: Settled the conjecture for all domains, completing the highly non-trivial algorithmic part!

“Polymorphisms beget algorithms”

“Lack of polymorphisms entails intractability”

How does one show “hardness”?

Computer scientists’ power tool: **Reductions**

“A reduces to B” :

if B can be solved efficiently, then so can A.

Contrapositive form:

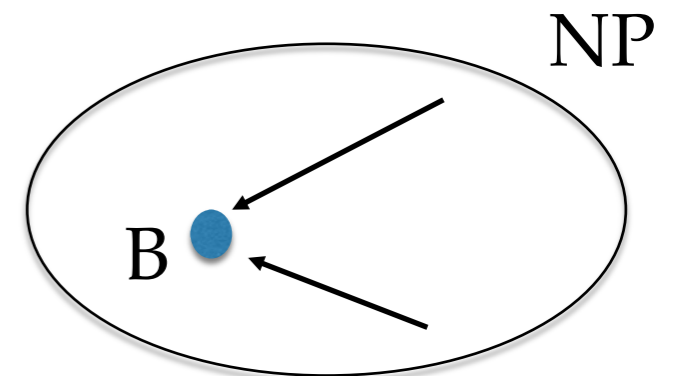
if A is (presumed) hard, then B must also be hard.

Problem B is **NP-hard**:

“All NP problems reduce to B.”

Equivalent to

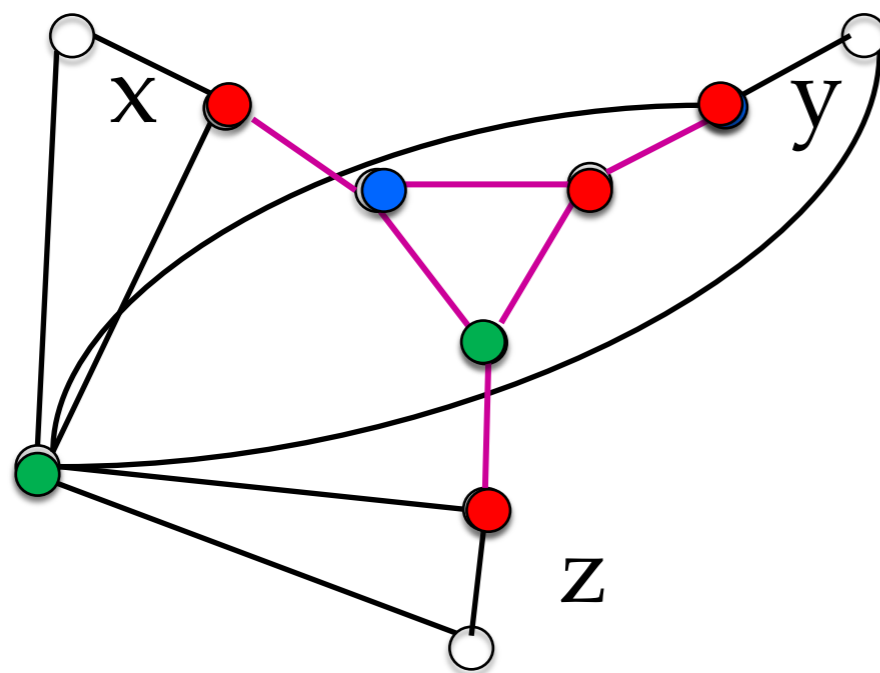
2-coloring 3-hypergraphs reduces to B



A “gadget” reduction

2-coloring 3-hypergraphs reduces to graph 3-coloring

Idea: Encode **Not-all-Equal**(x,y,z) by a *3-coloring graph gadget*



If x,y,z are not all equal,
can color using 3 colors

If x,y,z are equal,
can't color inner triangle

What do polymorphisms have to do with this?

Polymorphisms *characterize* such gadget reductions.

Local gadget reduction from $\text{CSP}(P)$ to $\text{CSP}(Q)$
if and only if P has “more” polymorphisms than Q .

Richer the polymorphisms, easier the problem

In particular if $\text{Pol}(Q)$ is “trivial” (only dictators)
then 2-coloring 3-hypergraphs reduces to $\text{CSP}(Q)$
 $\Rightarrow \text{CSP}(Q)$ is NP-hard

Polymorphisms and algorithms

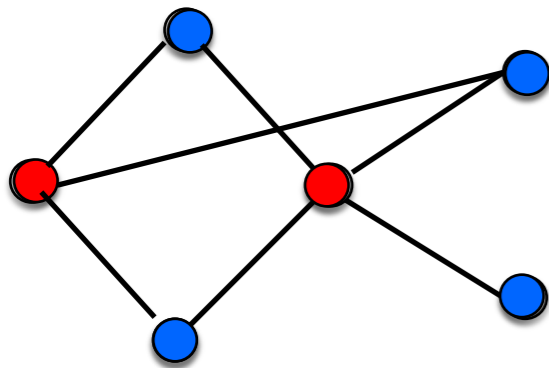
Polymorphisms can be composed with each other and thus have rich algebraic structure.

- Form a “clone” in universal algebraic terms

If $\text{Pol}(Q)$ includes some non-trivial polymorphism (with some weak form of symmetry), then it has more *structured* polymorphisms, which can be exploited to solve $\text{CSP}(Q)$

This is in general highly non-trivial, but I'll sketch the principle in a very simple case.

Polymorphisms and efficient 2-coloring algorithms



For $a, b \in \{0,1\}$, $a \neq b \Leftrightarrow a + b = 1$

Express 2-colorability as collection of equations of form $x_i + x_j = 1$ for each edge (i, j) in the graph ($x_i \in \{0,1\}$)

Linear program:

Stipulate $x_i \in [0,1]$
(Efficiently solvable)

“Rounding” algorithm:

$x_i > \frac{1}{2}$: Set $x_i = 1$
 $x_i < \frac{1}{2}$: Set $x_i = 0$

Works (if no $x_i = 1/2$)

Explained by
Majority polymorphism

Affine equations:

Stipulate $x_i \in \mathbb{Z}$
(Efficiently solvable)

“Rounding” algorithm:

$x_i > 0$: Set $x_i = 1$
 $x_i \leq 0$: Set $x_i = 0$

Works; explained by
Alternating Threshold
polymorphism

Linear system:

Stipulate $x_i \in \text{GF}(2)$,
field of 2 elements
(Efficiently solvable)

Works as is.

Explained by
Parity polymorphism

Polymorphisms and algorithms

2-coloring has abundance of polymorphisms:

Majority, AT, Parity,

leading to multitude of algorithms

Each of these classes *individually* enough to lead to algorithm.

[Brakensiek-G., '20]

Unified algorithm under family of *symmetric** polymorphisms

- Combination of linear program and affine equations
- Discovered in more general setting of “promise” CSPs

* $f(x_1, x_2, \dots, x_m) = f(x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(m)})$ for $\sigma \in S_m$

Actually block symmetry suffices

Beyond satisfiability of CSP

Suitable variants of polymorphisms govern tractability for extensions of CSP:

- Counting (count number of satisfying solutions)
- Optimization (satisfy as many constraints as possible)
- Approximation (estimate optimum value within some relative error)
- Fine-grained algorithms (for NP-hard CSPs, what's the best exponential runtime?)

Next up: **Promise CSP**

an exciting recent chapter in the

polymorphic gateway between

mathematical structure & algorithmic tractability

Relaxed form of coloring

NP-hard to 3-color a 3-colorable graph.

➤ What if I'm allowed to use **more** colors?

Can one efficiently 6-color a 3-colorable graph?

- No algorithm or NP-hardness known!
(most likely hard, some evidence exists, eg.
[G.-Sandeep'20] $O(1)$ -coloring hard under d-to-1 conjecture)
- 5-coloring NP-hard (shown only recently via
polymorphic framework [Barto-Bulin-Krokhin-Oprsal'20])
- Best known algorithm uses $\approx n^{0.19}$ colors!! 😞

Promise Constraint Satisfaction

[Austrin-G.-Håstad'17; Brakensiek-G.,'18]

Abstracting from coloring, since CSPs are typically hard, aim to solve a *relaxed* version of the constraints

Pair of predicates $P, Q : D^k \rightarrow \{0,1\}$

P **stricter than** Q (i. e., $P(x) \leq Q(x)$ for all $x \in D^k$)

$$\Phi_{strict} = P(x_1, x_2, x_4) \wedge P(x_2, x_3, x_7) \wedge P(x_1, x_5, x_6) \wedge \dots$$

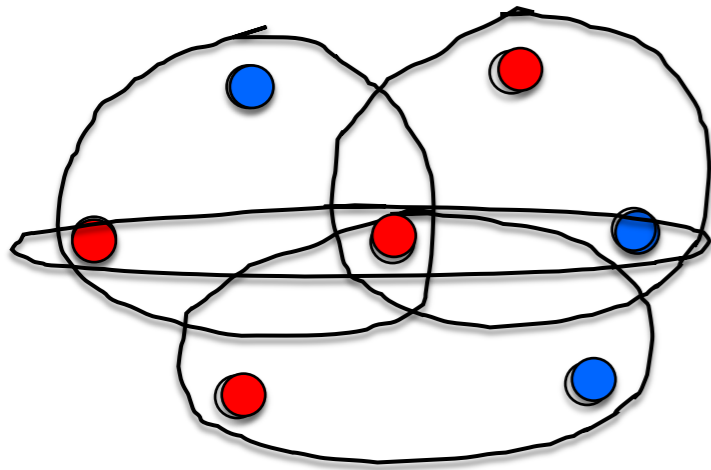
$$\Phi_{relaxed} = Q(x_1, x_2, x_4) \wedge Q(x_2, x_3, x_7) \wedge Q(x_1, x_5, x_6) \wedge \dots$$

Promise CSP (PCSP(P, Q)):

- Given that Φ_{strict} has a satisfying assignment, find an assignment satisfying $\Phi_{relaxed}$

CSP is special case when $P = Q$

Promise hypergraph coloring



NP-hard

2-coloring 3-hypergraph:

Assign vertices **Red** / **Blue** so that no triple is monochromatic.

What if we are *promised* the existence of a 2-coloring such that each triple has *exactly one Blue* ?

Can you now efficiently 2-color without monochromatic triple?

- Write down equation $x + y + z = 1$ for each triple.
- Solve system over *integers*.
- Color vertices with positive value **Blue**

Turns out, Yes!

- Has a *polymorphic* explanation (Alternating Threshold).
- This PCSP example has played an influential role in ensuing theory

Polymorphisms for PCSP

Let $P, Q: D^k \rightarrow \{0,1\}$ be predicates

$f: D^m \rightarrow D$ is a **polymorphism of P, Q** if whenever

$(x_1^{(1)}, x_2^{(1)}, \dots, x_k^{(1)})$ satisfies P

$(x_1^{(2)}, x_2^{(2)}, \dots, x_k^{(2)})$ satisfies P

\vdots

$(x_1^{(m)}, x_2^{(m)}, \dots, x_k^{(m)})$ satisfies P

$f \Downarrow \quad f \Downarrow \quad \dots \quad f \Downarrow$

(y_1, y_2, \dots, y_k) satisfies Q

Unlike for CSPs,
not closed under composition
(output satisfies
weaker predicate Q)

Theory much more complex;
must deal with
polymorphism *families*
rather than *individual* ones

Polymorphisms & PCSP: nascent theory

Limited polymorphisms \Rightarrow hardness

- e.g., hard if all polymorphisms depend only on *few* coordinates

Rich enough polymorphisms \Rightarrow efficient algorithms

- e.g., infinite family of *symmetric* polymorphisms

Unclear dividing line between “limited” and “rich enough”

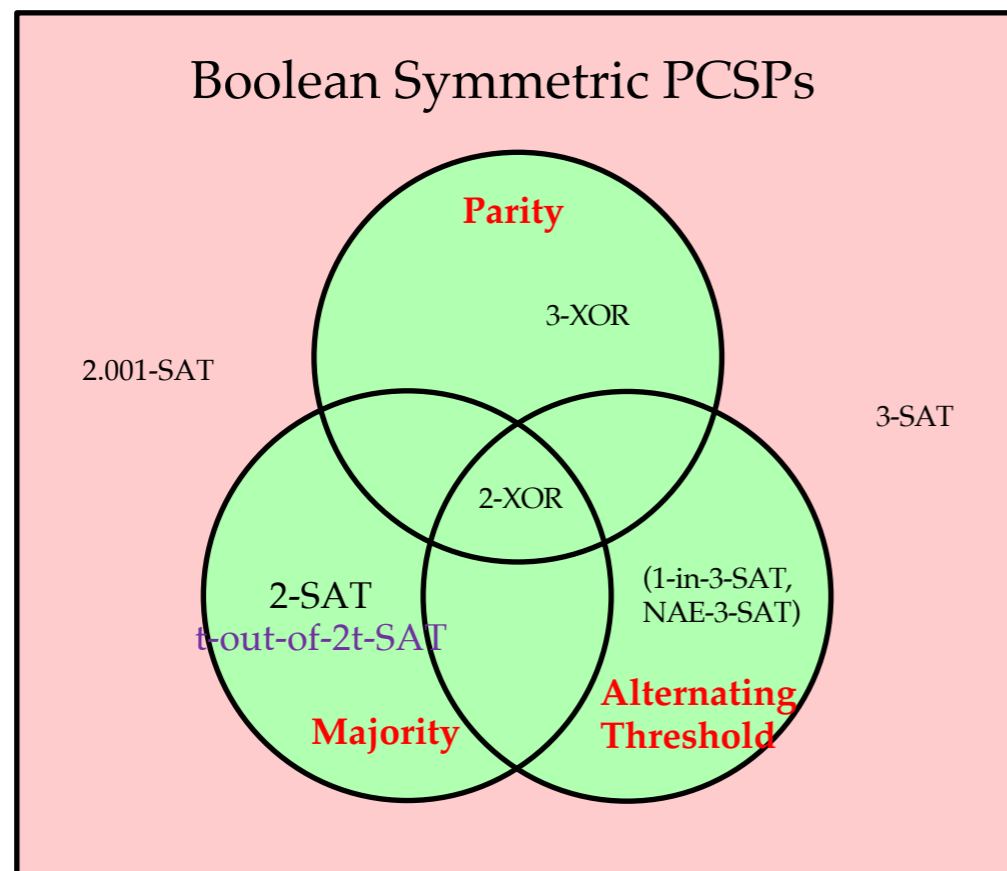
- Not even conjectured criterion
- Unlike CSPs, dichotomy might well not exist

Dichotomy for an interesting PCSP class

Boolean symmetric PCSP:

[Brakensiek-G.; Ficak-Kozik-Olsak-Stenkiewicz]

Every *Boolean* PCSP with *symmetric* predicates is polynomial time solvable or NP-hard.



■ Polynomial time algorithms

■ NP Hard

Dichotomy for an interesting PCSP class

Boolean symmetric PCSP:

Every *Boolean* PCSP with *symmetric* predicates is polynomial time solvable or NP-hard.

Algorithm: based on the polymorphisms
(doesn't rely on predicates' symmetry)

Hardness: Must lack Maj/AT/Parity of some odd arity (else have algo.)

- Then, using predicate symmetry, can show all polymorphisms depend essentially only on few coordinates
- Plugs into “PCP machinery” to yield hardness

Algorithm from symmetric polymorphisms

[Brakensiek-G.-Wrochna-Živný]

Efficient algorithm for PCSP(P,Q) when $\text{Pol}(P,Q)$ has an infinite family of *block-symmetric* polymorphisms.

- Combines basic linear program & affine equations (BLP+affine)

BLP+Affine*:

- Run the "Basic LP" algorithm. Reject if no solution.
- Run the "Affine" algorithm. Reject if no solution.
- Else, accept

* [Algo not quite this; ignoring a subtle issue...]

[Also, only solves decision problem,
search version open in general]

Basic LP Relaxation

Note: when $p_i(\cdot)$ and $w_{i,j}(\cdot)$ are in $\{0,1\}$, captures 2-SAT exactly.

Illustrate for 2-SAT

- Assign each to each variable / clause a probability distribution.
- Encoding the variable x_i :
Variables $p_i(0), p_i(1) \geq 0, p_i(0) + p_i(1) = 1$.
- Encoding a clause $x_i \vee x_j$:
Variables $w_{i,j}(1,0), w_{i,j}(0,1), w_{i,j}(1,1) \geq 0$, sum to 1.
- Consistency checks for marginals:
 $p_i(0) = w_{i,j}(0,1)$
 $p_i(1) = w_{i,j}(1,0) + w_{i,j}(1,1)$ and similarly for $p_j(0)$ and $p_j(1)$
- Solution over the rationals (if exists) can be found in polynomial time.

Affine Relaxation (for 2-SAT)

Note: when $q_i(\)$ and $z_{i,j}(\)$ are in $\{0,1\}$, captures 2-SAT exactly.

- Assign each to each variable / clause an affine combination.
- Encoding the variable x_i :
Variables $q_i(0), q_i(1) \in \mathbb{Z}, q_i(0) + q_i(1) = 1$.
- Encoding a clause $x_i \vee x_j$:
Variables $z_{i,j}(1,0), z_{i,j}(0,1), z_{i,j}(1,1) \in \mathbb{Z}$, sum to 1.
- Consistency checks:
 $q_i(0) = z_{i,j}(0,1)$
 $q_i(1) = z_{i,j}(1,0) + z_{i,j}(1,1)$ and similarly for $q_j(0)$ and $q_j(1)$
- Solution can be found in polynomial time. [Kannam, Bachem; 1979]

Analysis sketch (2-SAT)

Use Majority polymorphism (of sufficiently high odd arity) to argue *existence* of an assignment satisfying all clauses

x_i	x_j	say, $x_i \vee x_j$ is a clause
1	1	
⋮	⋮	$M w_{ij}(1,1) + m z_{i,j}(1,1)$ copies
1	1	
1	0	
⋮	⋮	$M w_{i,j}(1,0) + m z_{i,j}(1,0)$ copies
1	0	
0	1	
⋮	⋮	$M w_{i,j}(0,1) + m z_{i,j}(0,1)$ copies
0	1	

[polymorphism output]
(satisfies clause)

Pick integers $M \gg m$ s.t.

- $Mw(\cdot, \cdot) \in \mathbb{Z}^+$,
- $Mw(\cdot, \cdot) + m z(\cdot, \cdot) \geq 0$
- $M + m$ is odd

Key property:

consistent across clauses

$\text{Val}(x_i) = 1$ iff

$$M p_i(1) + m q_i(1) > (M + m)/2$$

New CSP Algorithm?

Exciting challenge: Can one find a streamlined algorithm combining linear and affine integer programs to solve all tractable cases of CSP?

BLP+Affine:

- Run the Basic LP algorithm. Reject if no solution.
- Run the Affine algorithm. Reject if no solution.
- Else, accept

BLP+Affine works for Boolean case but alas not in general

- fails for simple CSP on domain size 5 [Opršal]
- variant works for all CSPs over domains of size < 8 [Zhuk]

Polymorphic inspirations in optimization

LP algorithm for 2-coloring doesn't work
since there's the trivial solution that sets all $x_i = 1/2$

$$\begin{array}{l} x_i \in [0,1] \\ x_i + x_j = 1 \\ \text{for all edges } (i,j) \end{array}$$

Can we avoid variables getting value $1/2$?

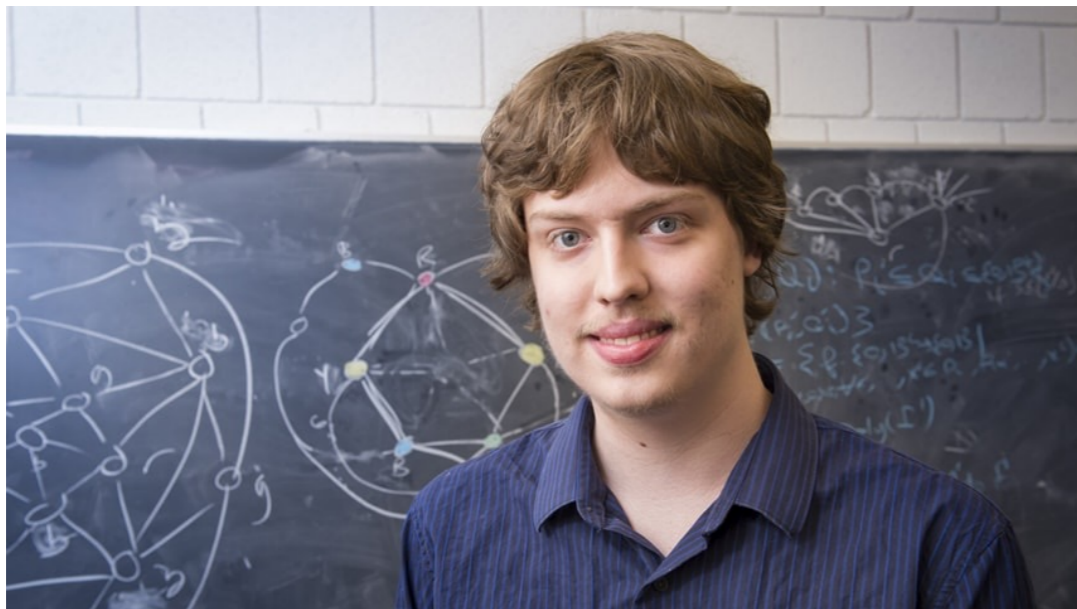
- Solve over a ring that avoids $1/2$, for example $\mathbb{Z}[\sqrt{2}]$?
 - This actually works! [Brakensiek-G., '19]
 - $(x, y) \mapsto (\sqrt{2} - 1)x + (2 - \sqrt{2})y$ is a polymorphism.
 - Polymorphic principle prescriptive of algorithm!

Acknowledgments

Per Austrin, Johan Håstad: $(2+\epsilon)$ -SAT is NP-hard

Libor Barto, Andrei Krokhin, Marcin Kozik,
Jakub Opršal, Dimitry Zhuk, Standa Živný

Special Thanks:



Joshua Brakensiek



Sai Sandeep

Summary

- Theory of computing: classify problems as easy / hard
- But *why* is something hard? *What* triggers tractability?
- *Polymorphic principle*: Polymorphisms \Rightarrow efficient algorithms
- Fully established for satisfiability of CSPs
 - also optimization, approximation, counting, etc.
- Nascent theory blending many tools for *promise* CSP
 - offers new insights and hopes for CSPs
 - many fascinating challenges
- Dream: deeper, systematic insight into P ?

Thank you for your attention!