Proof complexity of the graph isomorphism problem

joint work with Albert Atserias

Joanna Ochremiak

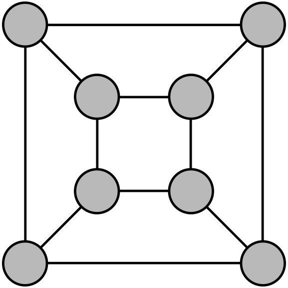
CNRS, LaBRI

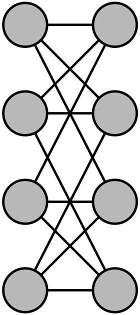
Journées Nationales de l'Informatique Mathématique
13th March 2019

The graph isomorphism problem

Input: graphs G and H

Question: are G and H isomorphic?

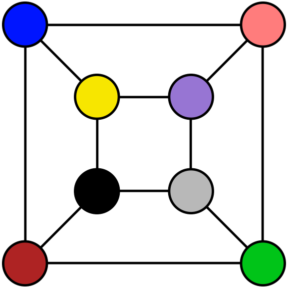


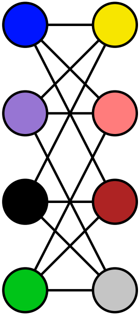


The graph isomorphism problem

Input: graphs G and H

Question: are G and H isomorphic?





Complexity

not likely to be NP-complete

The graph isomorphism problem is in NP.

Not known to be solvable in polynomial time.

Theorem [Babai'17]. The graph isomorphism problem is solvable in time $2^{O(log(n)^c)}$, for some fixed $c > 0$.

best we know

This talk

What is the power of such algorithms?
Which instances can we solve?

Algorithms that compute:

- an answer and
- a certificate/proof that the answer is correct.

Approach: Study algorithms by analysing proof systems.

Compare with:

- combinatorial algorithms
- distinguishability in logic

Algebraic and mathematical-programming techniques

Step 1: encode an instance as a system of equations,

Step 2: solve the system.

Algebraic and mathematical-programming techniques

Step 1: encode an instance as a system of equations,

~~Step 2: solve the system.~~

Step 2: determine if there EXISTS a solution.

We only want to know if there EXISTS an isomorphism.

Step 1: equations

Input: graphs G and H

Compute: a system of equations $\mathrm{ISO}(G,H)$

$$\begin{cases} x_{vw}^2 - x_{vw} = 0 & \text{for every } v \in V(G), w \in V(H) \\ \sum_{w \in V(H)} x_{vw} - 1 = 0 & \text{for every } v \in V(G) \\ \sum_{v \in V(G)} x_{vw} - 1 = 0 & \text{for every } w \in V(H) \\ x_{vw}x_{v'w'} = 0 & \text{if } (v,v') \in E(G), (w,w') \notin E(H) \\ x_{vw}x_{v'w'} = 0 & \text{if } (v,v') \notin E(G), (w,w') \in E(H) \end{cases}$$

$$\text{SOLUTION} \iff \text{ISOMORPHISM}$$

Solving systems of polynomial equations is intractable.

Algebraic and mathematical-programming techniques

Step 1: encode an instance as a system of equations,

~~Step 2: solve the system.~~

~~Step 2: determine if there exists a solution.~~

Step 2: APPROXIMATELY determine if there exists a solution.

We can use proof systems!

Step 2: computing a proof

Step 2: compute a proof that there is no solution

always correct

Output:

- if the algorithm finds a proof $\rightarrow$ *"no isomorphism"*
- otherwise $\rightarrow$ *"I do not know"*

Which pairs of non-isomorphic graphs the algorithms DISTINGUISH?

output *"no isomorphism"*

Proofs

Step 2: compute a proof that there is no solution

different **type of proof** $\leftrightarrow$ different algorithm

Algorithms:

- linear programming
- Gröbner basis
- semidefinite programming

Semidefinite Proofs

$$\begin{cases} x^2 + y + 2 = 0 \\ x - y^2 + 3 = 0 \end{cases}$$

$$-6\cdot(x^2 + y + 2) + 2\cdot(x - y^2 + 3) + \frac{1}{3} + 2\left(y + \frac{3}{2}\right)^2 + 6\left(x - \frac{1}{6}\right)^2 = -\mathbf{1}$$

Semidefinite Proofs

$$\begin{cases} x^2 + y + 2 = 0 \\ x - y^2 + 3 = 0 \end{cases}$$

A semidefinite proof that there is no solution:

$$-6\cdot(x^2 + y + 2) + 2\cdot(x - y^2 + 3) + \frac{1}{3} + 2\left(y + \frac{3}{2}\right)^2 + 6\left(x - \frac{1}{6}\right)^2 = -\mathbf{1}$$

Semidefinite Proofs

$$\begin{cases} x^2 + y + 2 = 0 \\ x - y^2 + 3 = 0 \end{cases}$$

A semidefinite proof that there is no solution:

$$-6\cdot(x^2 + y + 2) + 2\cdot(x - y^2 + 3) + \frac{1}{3} + 2\left(y + \frac{3}{2}\right)^2 + 6\left(x - \frac{1}{6}\right)^2 = -\mathbf{1}$$

arbitrary polynomials

sum of squares of polynomials

Semidefinite Proofs

$$\begin{cases} x^2 + y + 2 = 0 \\ x - y^2 + 3 = 0 \end{cases}$$

A semidefinite proof that there is no solution:

$$-6 \cdot (x^2 + y + 2) + 2 \cdot (x - y^2 + 3) + \frac{1}{3} + 2\left(y + \frac{3}{2}\right)^2 + 6\left(x - \frac{1}{6}\right)^2 = -\mathbf{1}$$

arbitrary polynomials

sum of squares of polynomials

degree of the proof $\to$ max degree of polynomials on the left

Finding Semidefinite Proofs

$$\begin{cases} x^2 + y + 2 = 0 \\ x - y^2 + 3 = 0 \end{cases}$$

A semidefinite proof **of degree 2** that there is no solution:

Finding Semidefinite Proofs

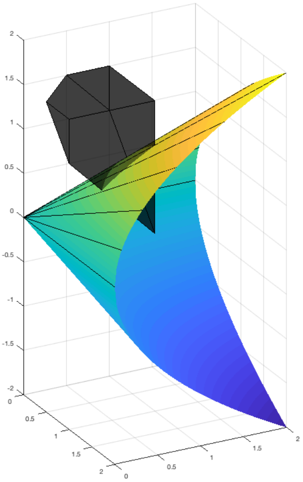
$$\begin{cases} x^2 + y + 2 = 0 \\ x - y^2 + 3 = 0 \end{cases}$$

A semidefinite proof **of degree 2** that there is no solution:

$$a \cdot (x^2 + y + 2) + b \cdot (x - y^2 + 3) + cx^2 + dy^2 + exy + fx + gy + h = -\mathbf{1}$$

sum of squares of polynomials

Finding Semidefinite Proofs



Proofs

Step 2: compute a proof that there is no solution

restriction of semidefinite programming
finding a proof: linear inequalities

Algorithms:

- linear programming
- Gröbner basis
- semidefinite programming

computing a generating set
in the ideal of polynomials

Proofs

Step 2: compute a proof that there is no solution

~~Algorithms:~~ Techniques:

- linear programming **hierarchy of algorithms**
- Gröbner basis **hierarchy of algorithms**
- semidefinite programming **hierarchy of algorithms**

degree of polynomials
in the proof

Summary of the setting

Algebraic and mathematical-programming techniques:

Step 1: encode an instance as a system of equations,

Step 2: compute a proof that there is no solution

always correct

Output:

- if the algorithm finds a proof $\rightarrow$ *"no isomorphism"*
- otherwise $\rightarrow$ *"I do not know"*

Which pairs of non-isomorphic graphs the algorithms DISTINGUISH?

output *"no isomorphism"*

Colour refinement algorithm

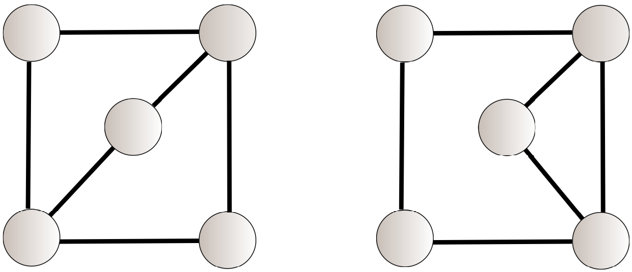
1. take $G \dot\cup H$
2. assign the same colour to all vertices

Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round

Colour refinement algorithm

1. take $G \mathbin{\dot\cup} H$
2. assign the same colour to all vertices

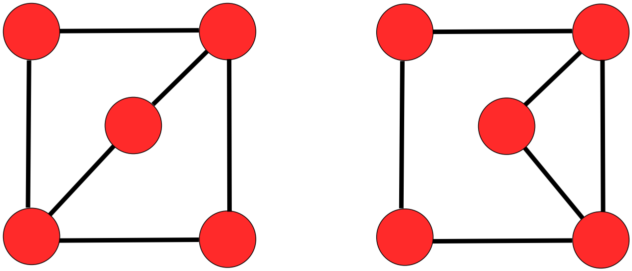
Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round



Colour refinement algorithm

1. take $G \dot\cup H$
2. assign the same colour to all vertices

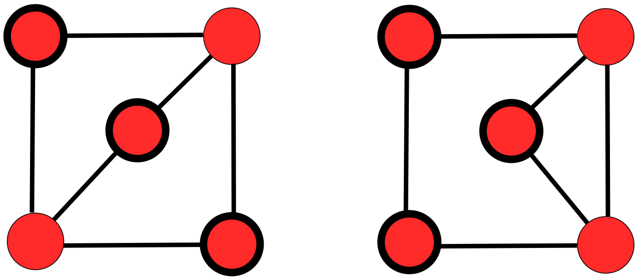
Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round



Colour refinement algorithm

1. take $G \dot\cup H$
2. assign the same colour to all vertices

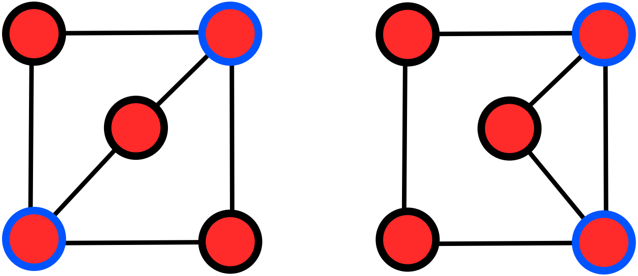
Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round



Colour refinement algorithm

1. take $G \dot\cup H$
2. assign the same colour to all vertices

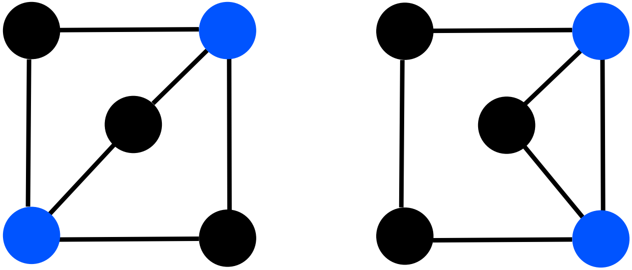
Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round



Colour refinement algorithm

1. take $G \,\dot{\cup}\, H$
2. assign the same colour to all vertices

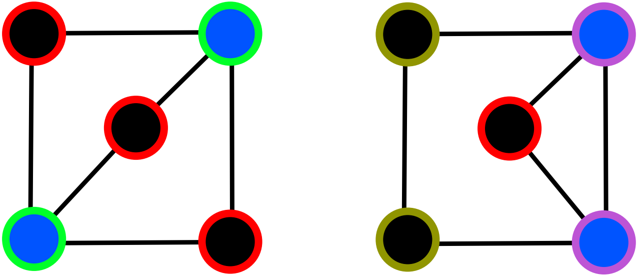
Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round



Colour refinement algorithm

1. take $G \dot\cup H$
2. assign the same colour to all vertices

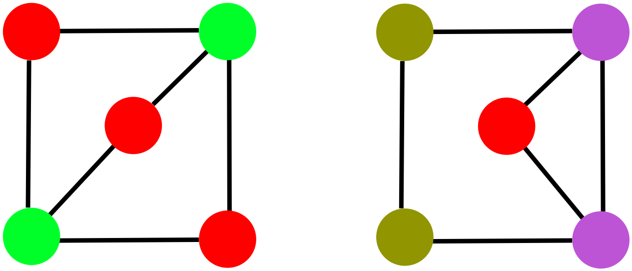
Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round



Colour refinement algorithm

1. take $G \dot\cup H$
2. assign the same colour to all vertices

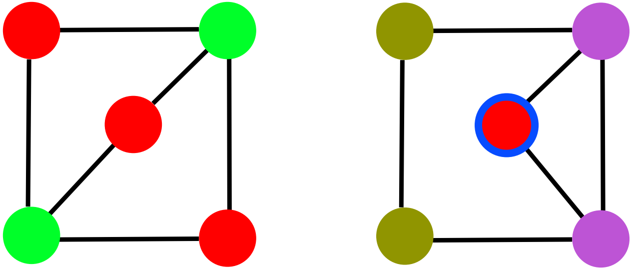
Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round



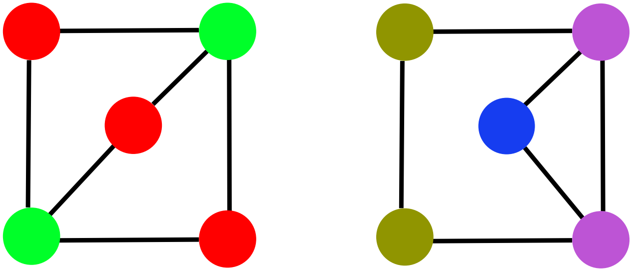
Colour refinement algorithm

1. take $G \dot\cup H$
2. assign the same colour to all vertices

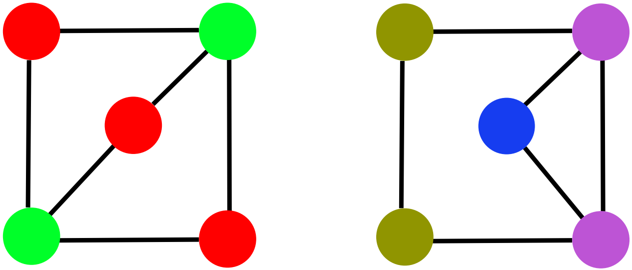
Iterate: assign different colours to vertices that have a different number of neighbours of at least one colour assigned in the previous round



Colour refinement algorithm



Colour refinement algorithm



the number of vertices of some colour in G is different than the number of vertices of this colour in H $\rightarrow$ *"no isomorphism"*

colourings are the same $\rightarrow$ *"I do not know"*

k-dimensional Weisfeiler-Lehman algorithm

Similar but we colour k-tuples of vertices :-)

Counting logic

$C^k_{\infty\omega}$ - first-order logic with:

- counting quantifiers $\exists^{\geq m}$
- infinite disjunctions and conjunctions
- at most k variables

$\forall x\big((\exists^{\geq d} y\, E(x,y)) \wedge (\neg\exists^{\geq d+1} y\, E(x,y))\big)$ - graph is d-regular

k-WL and counting logic

The counting logic $C^2_{\infty\omega}$ distinguishes G and H.

$\Updownarrow$ **[Immerman, Lander'90]**

Colour refinement distinguishes G and H.

The counting logic $C^{k+1}_{\infty\omega}$ distinguishes G and H.

$\Updownarrow$ **[Cai, Fūrer, Immerman'92]**

k-dimensional Weisfeiler-Lehman algorithm distinguishes G and H.

Correspondence

Theorem [Atserias, Maneva'13] [Malkin'14] [Grohe, Otto'11] [Berkholz, Grohe'15].

The counting logic $C^{k+1}_{\infty\omega}$ distinguishes G and H.

$$\Updownarrow$$

k-dimensional Weisfeiler-Lehman algorithm distinguishes G and H.

$$\Updownarrow$$

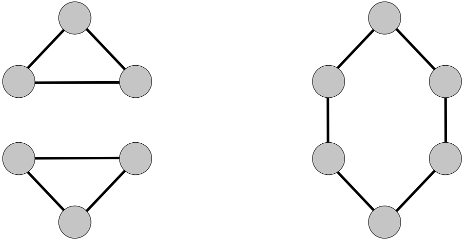
Linear programming degree $k+1$ distinguishes G and H.

Consequences

Theorem [Babai, Kučera'80]. Linear programming degree 2 distinguishes almost all graphs.

outputs *"I do not know"*

It does not distinguish:



Relative power

For every pair of non-isomorphic graphs G and H:

Linear programming degree k distinguishes G and H.

$\Downarrow$ **[Berkholz, Grohe'15]**

Gröbner basis degree k distinguishes G and H.

$\Downarrow$ **[Berkholz'18]**

Semidefinite programming degree $2k$ distinguishes G and H.

Relative power

For every pair of non-isomorphic graphs G and H:

Linear programming degree k distinguishes G and H.

$\Downarrow$ **[Berkholz, Grohe'15]**

Gröbner basis degree k distinguishes G and H.

$\Downarrow$ **[Berkholz'18]**

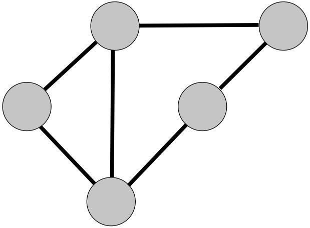
Semidefinite programming degree $2k$ distinguishes G and H.

Does semidefinite programming distinguish more graphs than linear programming?

Hope: yes!

Semidefinite programming much more powerful for many problems.

Example: MAX CUT

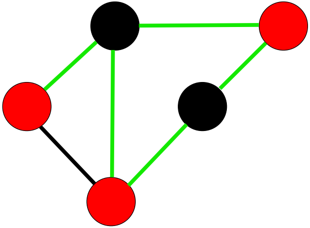


Semidefinite programming: best known efficient approximation
Linear programming: very bad approximation

Hope: yes!

Semidefinite programming much more powerful for many problems.

Example: MAX CUT



Semidefinite programming: best known efficient approximation
Linear programming: very bad approximation

All algorithms are equally powerful!

For every pair of non-isomorphic graphs G and H:

Linear programming degree k distinguishes G and H.

$\Downarrow$ **[Berkholz, Grohe'15]**

Gröbner basis degree k distinguishes G and H.

$\Downarrow$ **[Berkholz'18]**

Semidefinite programming degree $2k$ distinguishes G and H.

$\Downarrow$ **[Atserias, O.'18]**

Linear programming degree ck distinguishes G and H.

constant independent from k

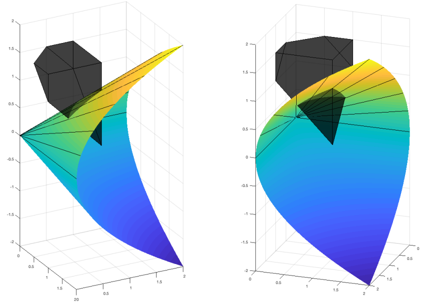
All algorithms are equally powerful!

Theorem. For the graph isomorphism problem all three algorithmic techniques are equally powerful, up to a constant factor loss in the degree.

Proof

Fact. Existence of semidefinite proofs reduces to feasibility of SDPs.

Is *polytop* $\cap$ *cone of positive semidefinite matrices* non-empty?



Key: There exists c, such that feasibility of SDPs is expressible in the counting logic $C^{c}_{\infty\omega}$.

Proof

For every pair of non-isomorphic graphs G and H:

Semidefinite programming degree $2k$ distinguishes G and H.

$$\Downarrow$$

The counting logic $C^{ck}_{\infty\omega}$ distinguishes G and H.

$$\Updownarrow$$

Linear programming degree ck distinguishes G and H.