Is Ryser's formula for the Permanent optimal?

Holger Dell

Dániel Marx

Thore Husfeldt

Nina Taslaman

Martin Wahlén

The Permanent of a $d \times d$ matrix A

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)}$$

The Permanent of a $d \times d$ matrix A

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)}$$

Ryser's formula (1963)

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j}$$

The Permanent of a $d \times d$ matrix A

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)}$$ has $d! \leq 2^{d \log d}$ terms.

Ryser's formula (1963)

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j}$$

The Permanent of a $d \times d$ matrix A

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)}$$ has $d! \leq 2^{d \log d}$ terms.

Ryser's formula (1963)

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j}$$ has 2^d terms.

The Permanent of a $d \times d$ matrix A

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)}$$ has $d! \leq 2^{d \log d}$ terms.

Ryser's formula (1963)

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j}$$ has 2^d terms.

Valiant (1997): No $\mathrm{poly}(d)$-size formula unless P=NP.

The Permanent of a $d \times d$ matrix A

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)} \quad \text{has } d! \leq 2^{d \log d} \text{ terms.}$$

Ryser's formula (1963)

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j} \quad \text{has } 2^d \text{ terms.}$$

Valiant (1997): No $\mathrm{poly}(d)$-size formula unless P=NP.

We (2013+): No $2^{o(d)}$-size formula unless "ETH is false".

The Permanent of a $d \times d$ matrix A

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)} \quad \text{has } d! \leq 2^{d \log d} \text{ terms.}$$

Ryser's formula (1963)

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j} \quad \text{has } 2^d \text{ terms.}$$

Valiant (1997): No $\mathrm{poly}(d)$-size formula unless P=NP.

We (2013+): No $2^{o(d)}$-size formula unless "ETH is false".

Do 1.99^d-size formulas exist?

The Permanent of a $d \times d$ matrix A

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)}$$ has $d! \leq 2^{d \log d}$ terms.

Ryser's formula (1963)

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j}$$ has 2^d terms.

Valiant (1997): No $\mathrm{poly}(d)$-size formula unless P=NP.

We (2013+): No $2^{o(d)}$-size formula unless "ETH is false".

Do 1.99^d-size formulas exist?

(we mean uniform formulas here)

The Exponential Time Hypothesis

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

[Many] 3-SAT **is** computable in time 2^n.

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

[Many] 3-SAT **is** computable in time 1.334^n.

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

[Many] 3-SAT **is** computable in time 1.324^n.

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

[Many] 3-SAT **is** computable in time 1.32113^n.

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

[Many] 3-SAT **is** computable in time c^n for certain $c > 1$.

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

[Many] 3-SAT **is** computable in time c^n for certain $c > 1$.

Exponential Time Hypothesis (ETH):
3-SAT is **not** computable in time 1.0001^n.

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

[Many] 3-SAT **is** computable in time c^n for certain $c > 1$.

Exponential Time Hypothesis (ETH):
3-SAT is **not** computable in time c^n for some $c > 1$.

The Exponential Time Hypothesis

3-SAT : Satisfiability of Boolean 3-CNF formulas

Given $\varphi = \left(x \vee y \vee \bar{z}\right) \wedge \left(\bar{x} \vee \bar{y} \vee z\right) \wedge \left(\bar{x} \vee y \vee z\right) \wedge \ldots$

Compute whether φ is satisfiable or not.

[Many] 3-SAT **is** computable in time c^n for certain $c > 1$.

Exponential Time Hypothesis (ETH):
3-SAT is **not** computable in time c^n for some $c > 1$.

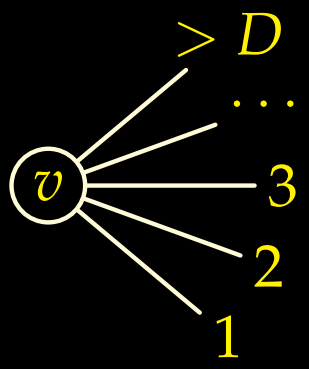
Sparsification Lemma. [Impagliazzo, Paturi, Zane '01]
ETH $\Leftrightarrow$ k-SAT is not computable in time $2^{o(\#\text{clauses})}$.

Sparsification of Vertex Cover

(implicit in [Johnson, Szegedy ’98])

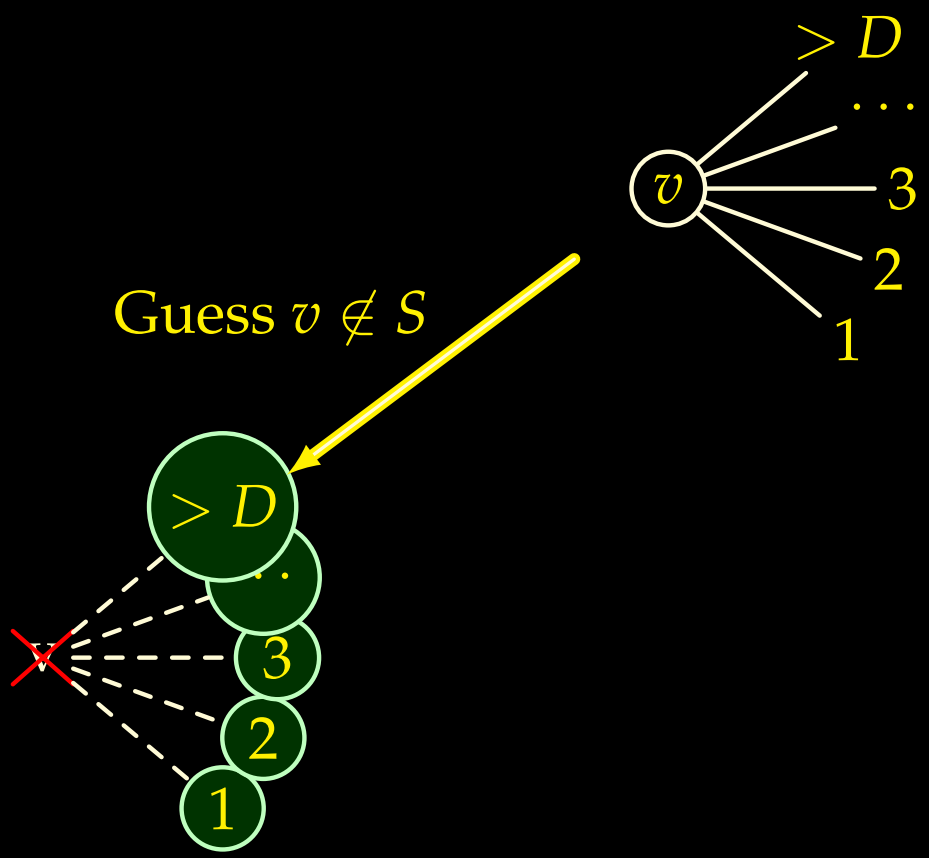
Sparsification of Vertex Cover

(implicit in [Johnson, Szegedy '98])



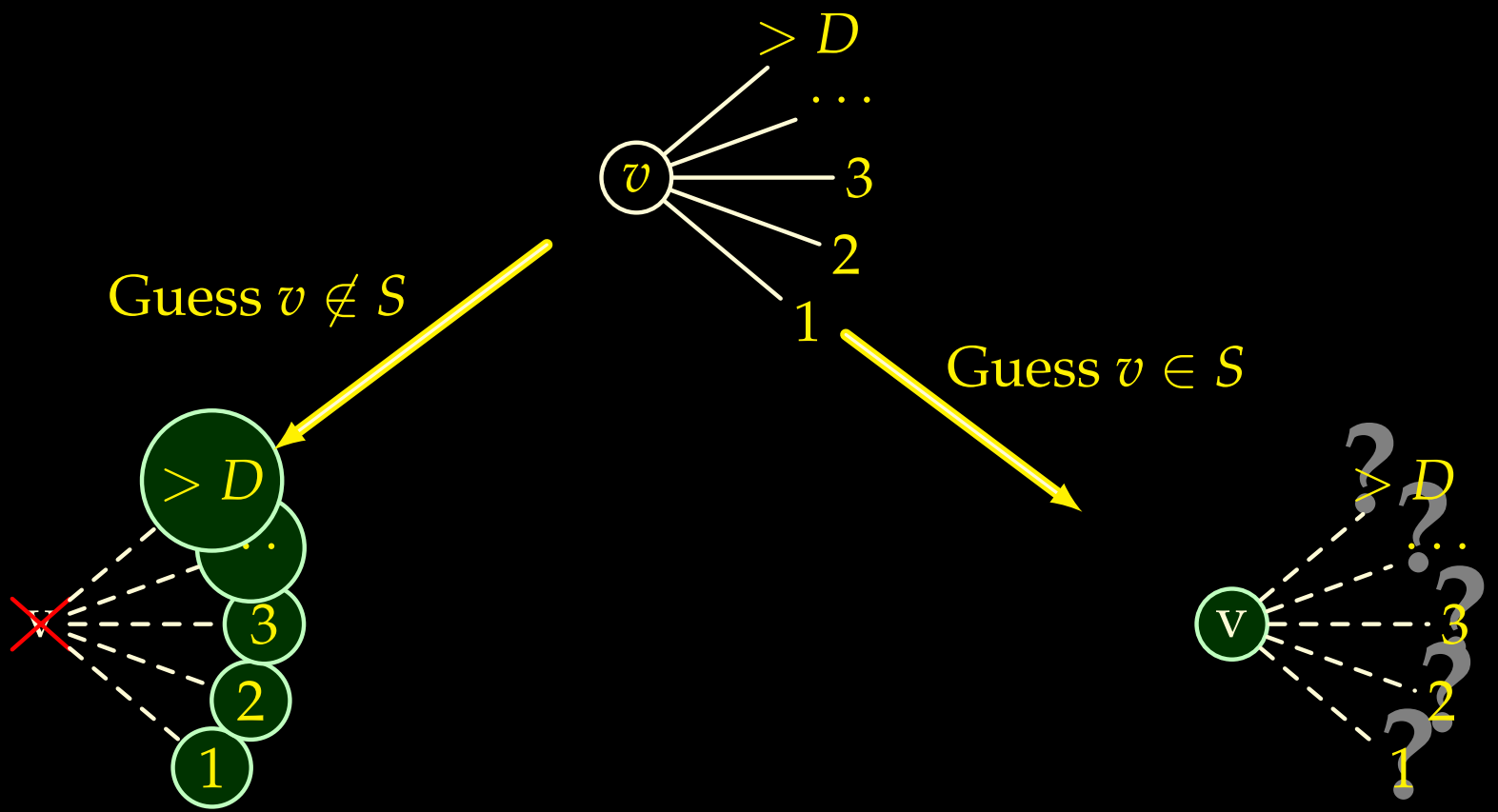
Sparsification of Vertex Cover

(implicit in [Johnson, Szegedy '98])

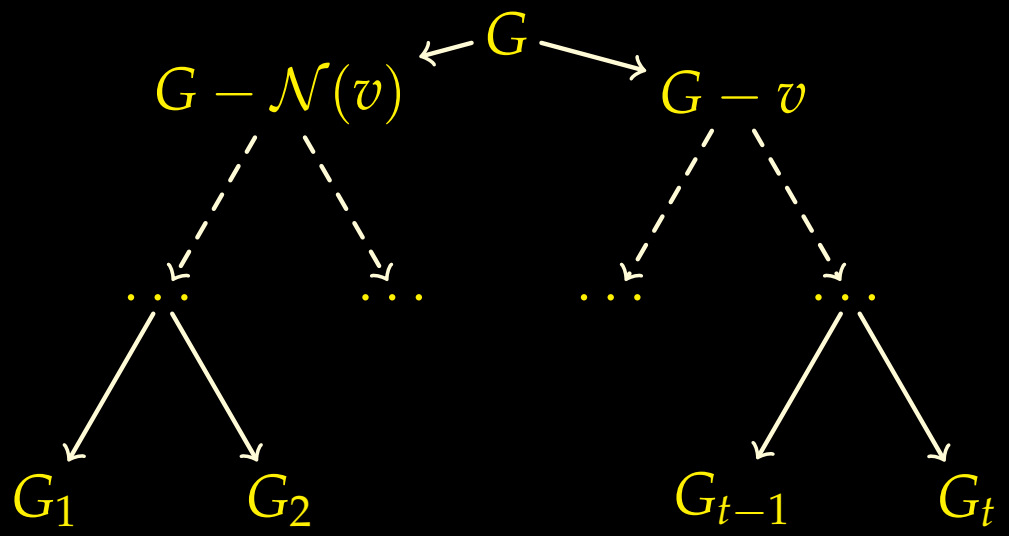


Sparsification of Vertex Cover

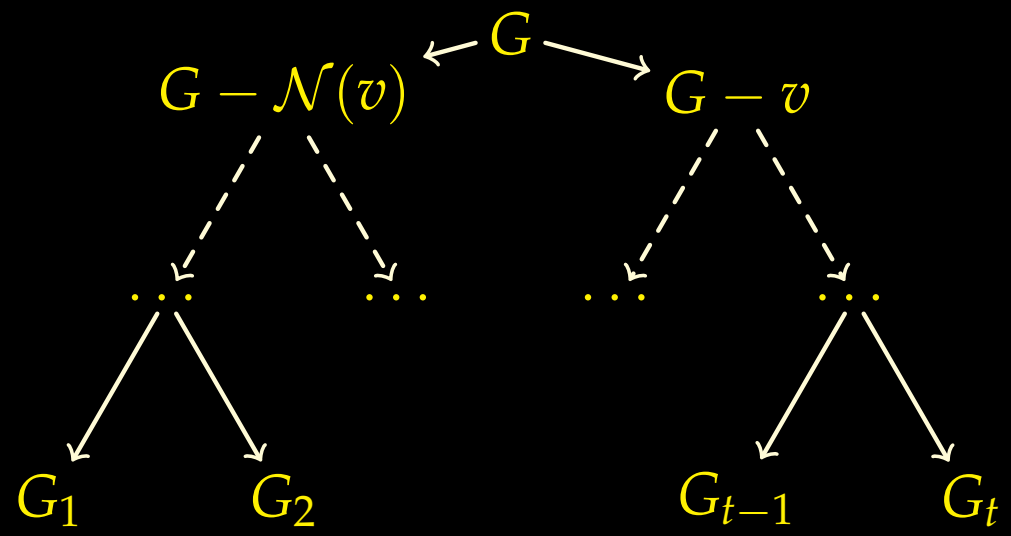
(implicit in [Johnson, Szegedy '98])



Sparsification of Vertex Cover II

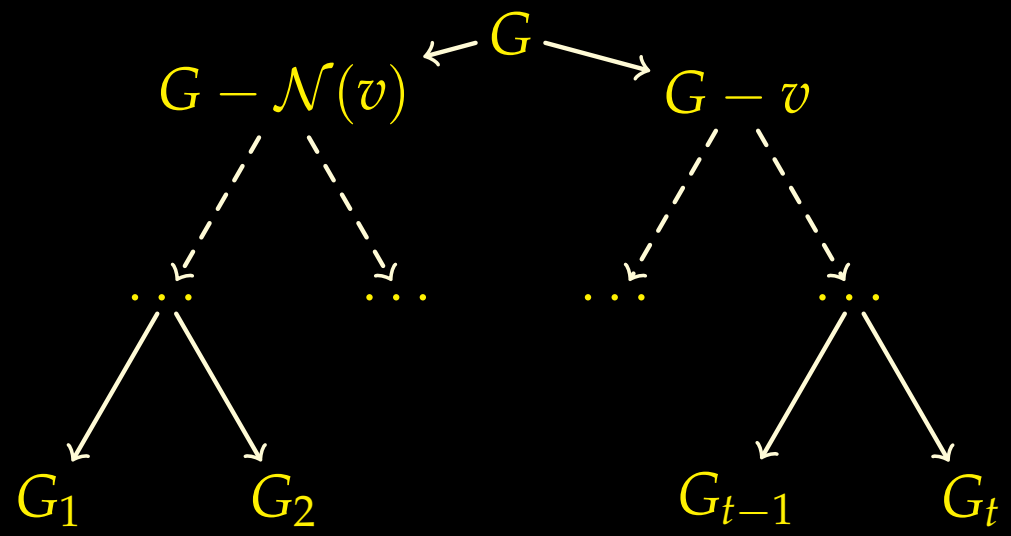


Sparsification of Vertex Cover II



$$T(n) \leq T(n-D) + T(n-1) \leq 2^{h(1/D)n}$$

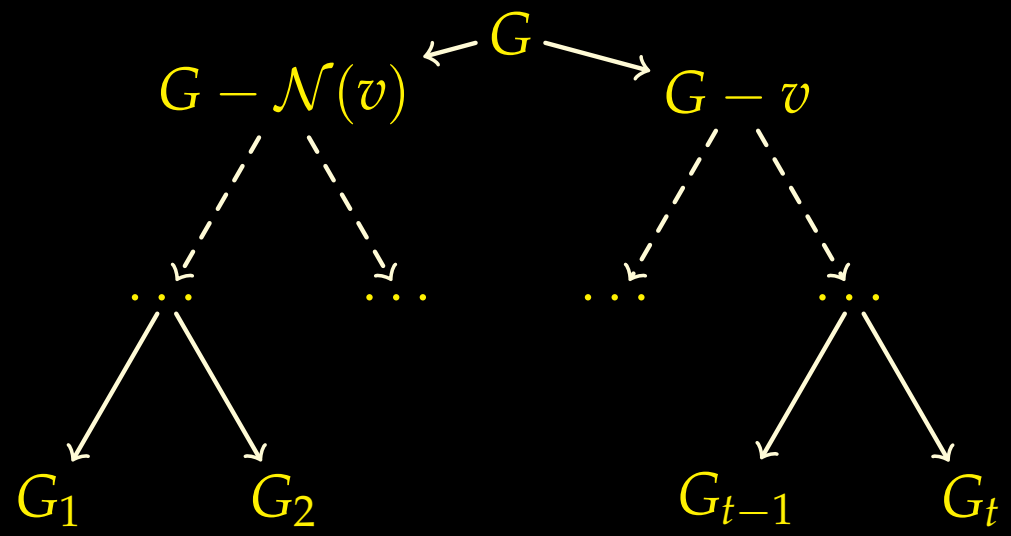
Sparsification of Vertex Cover II



$$T(n) \leq T(n-D) + T(n-1) \leq 2^{h(1/D)n}$$

If Vertex Cover is solvable in time $2^{o(m)}$,

Sparsification of Vertex Cover II

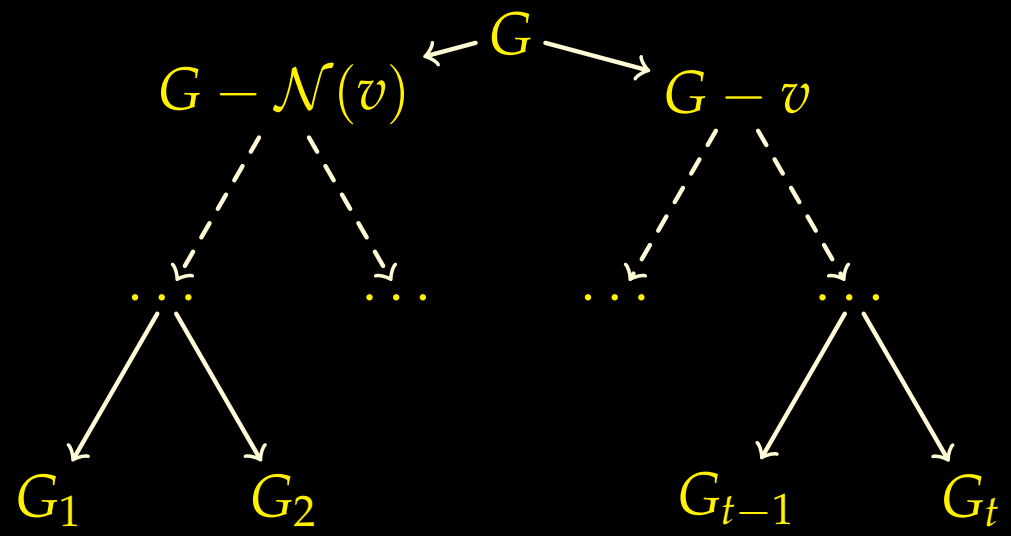


$$T(n) \leq T(n-D) + T(n-1) \leq 2^{h(1/D)n}$$

If Vertex Cover is solvable in time $2^{o(m)}$,

then it is solvable in time $t \cdot 2^{o(m(G_i))}$

Sparsification of Vertex Cover II

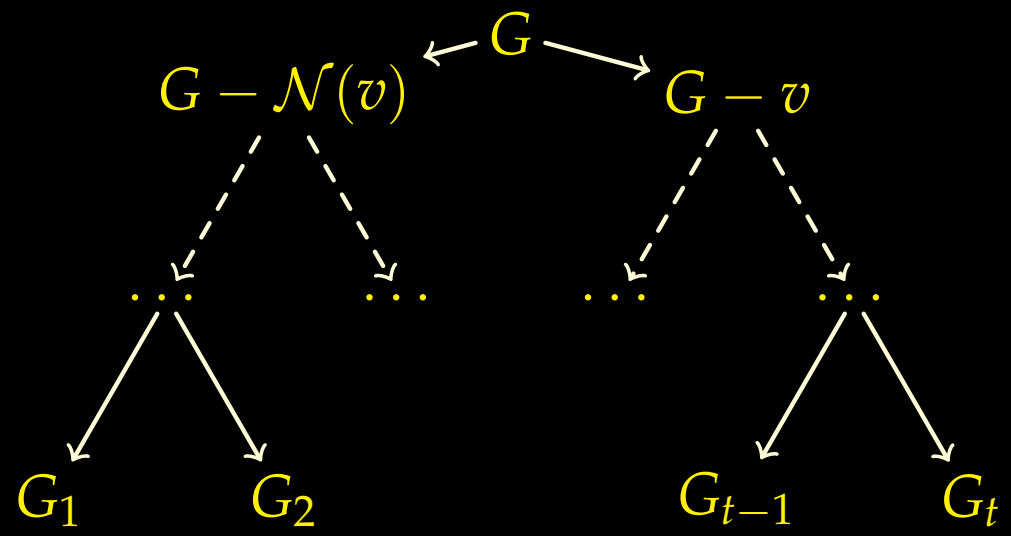


$$T(n) \leq T(n-D) + T(n-1) \leq 2^{h(1/D)n}$$

If Vertex Cover is solvable in time $2^{o(m)}$,

then it is solvable in time $t \cdot 2^{o(m(G_i))} \leq 2^{h(1/D)n} \cdot 2^{o(Dn)}$

Sparsification of Vertex Cover II



$$T(n) \leq T(n-D) + T(n-1) \leq 2^{h(1/D)n}$$

If Vertex Cover is solvable in time $2^{o(m)}$,

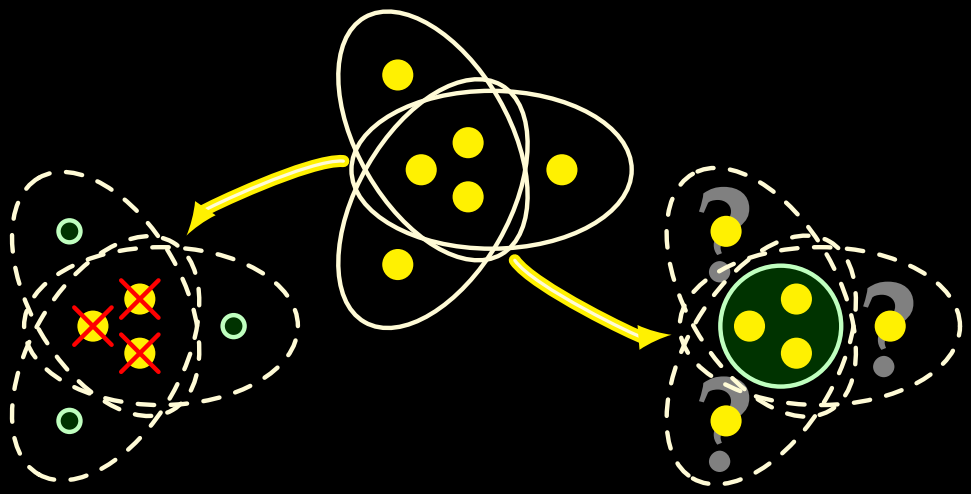
then it is solvable in time $t \cdot 2^{o(m(G_i))} \leq 2^{h(1/D)n} \cdot 2^{o(Dn)} \leq 2^{o(n)}$.

The Sparsification Lemma

Theorem. [Impagliazzo, Paturi, Zane '01]
Map k-CNF φ to few sparse subformulas $\varphi_1, \ldots, \varphi_t$

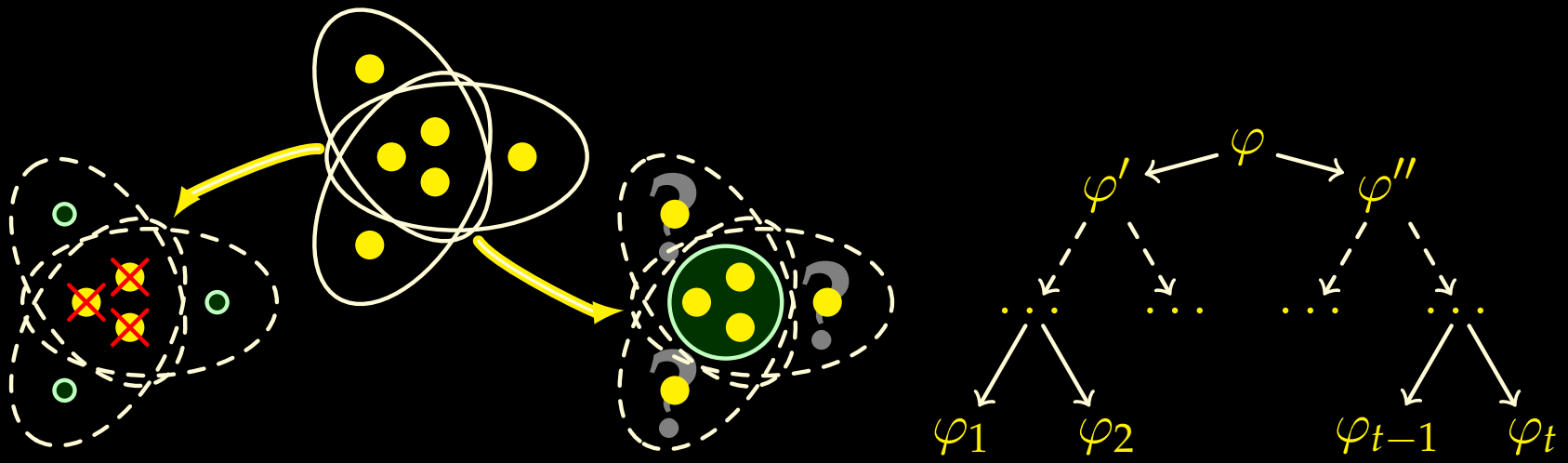
The Sparsification Lemma

Theorem. [Impagliazzo, Paturi, Zane '01]
Map k-CNF φ to few sparse subformulas $\varphi_1, \ldots, \varphi_t$



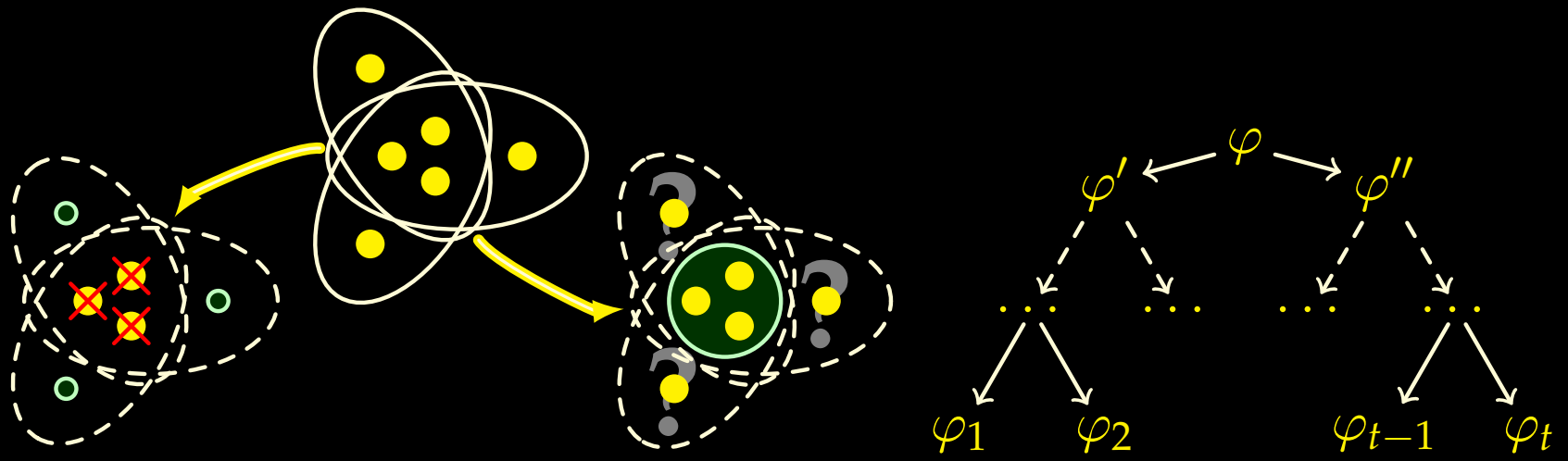
The Sparsification Lemma

Theorem. [Impagliazzo, Paturi, Zane '01]
Map k-CNF φ to few sparse subformulas $\varphi_1, \ldots, \varphi_t$



The Sparsification Lemma

Theorem. [Impagliazzo, Paturi, Zane '01]
Map k-CNF φ to few sparse subformulas $\varphi_1, \ldots, \varphi_t$



such that $\mathrm{sat}(\varphi) = \mathrm{sat}(\varphi_1) \dot\cup \ldots \dot\cup \mathrm{sat}(\varphi_t)$.

Next...

ETH is false if the permanent can be computed in time $2^{o(d)}$.

Reduction: 3-SAT to Permanent

Reduction: 3-SAT to Permanent

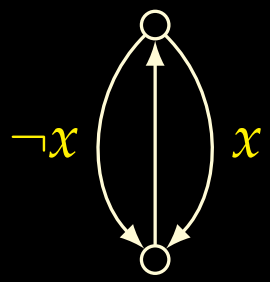
Given a Boolean formula φ with m clauses,
construct a $(d \times d)$-matrix A with dimension $d \leq O(m)$.

Reduction: 3-SAT to Permanent

Given a Boolean formula φ with m clauses,
construct a $(d \times d)$-matrix A with dimension $d \leq O(m)$.
A is the adjacency matrix of a **graph** with these gadgets:

Reduction: 3-SAT to Permanent

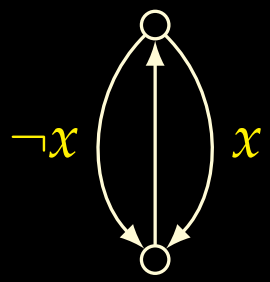
Given a Boolean formula φ with m clauses,
construct a $(d \times d)$-matrix A with dimension $d \leq O(m)$.
A is the adjacency matrix of a **graph** with these gadgets:



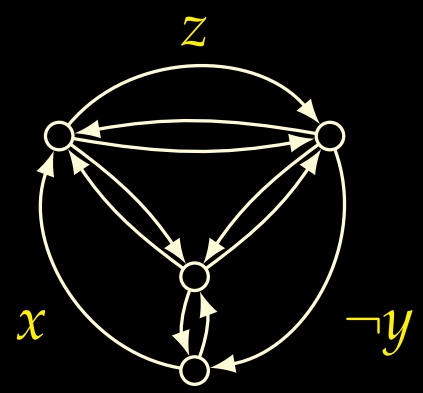
variable x

Reduction: 3-SAT to Permanent

Given a Boolean formula φ with m clauses,
construct a $(d \times d)$-matrix A with dimension $d \leq O(m)$.
A is the adjacency matrix of a **graph** with these gadgets:



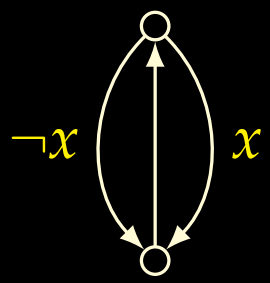
variable x



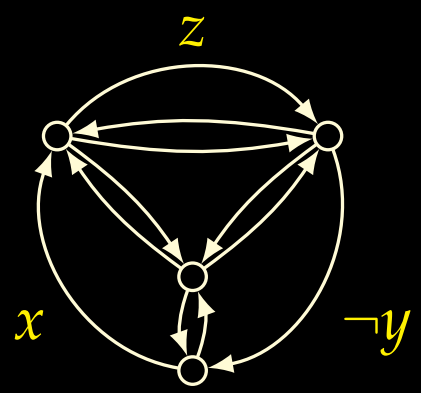
clause $(\neg x \vee y \vee \neg z)$

Reduction: 3-SAT to Permanent

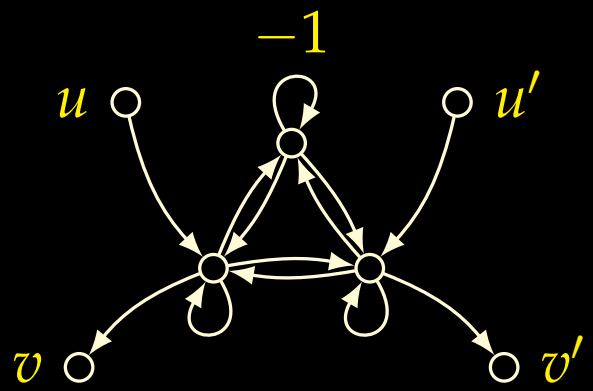
Given a Boolean formula φ with m clauses,
construct a $(d \times d)$-matrix A with dimension $d \leq O(m)$.
A is the adjacency matrix of a **graph** with these gadgets:



variable x



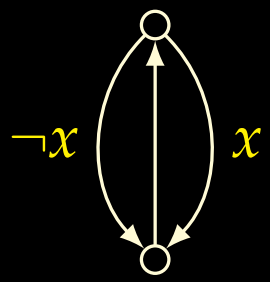
clause $(\neg x \vee y \vee \neg z)$



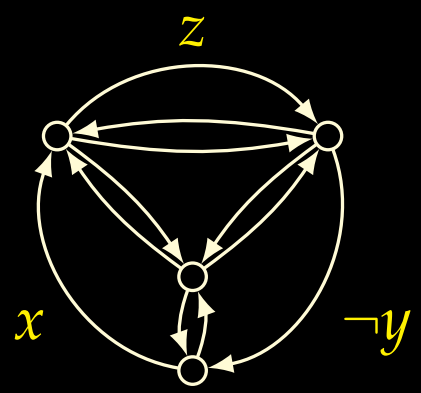
"(uv) equals $(u'v')$"

Reduction: 3-SAT to Permanent

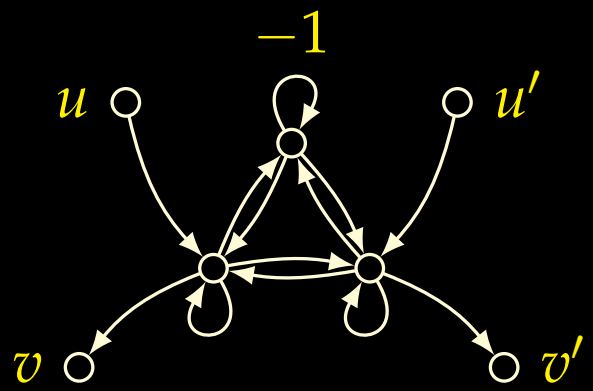
Given a Boolean formula φ with m clauses,
construct a $(d \times d)$-matrix A with dimension $d \leq O(m)$.
A is the adjacency matrix of a **graph** with these gadgets:



variable x



clause $(\neg x \vee y \vee \neg z)$



"(uv) equals $(u'v')$"

$$\mathrm{Perm}(A) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi).$$

Reduction to the 01-Permanent

$$\mathrm{Perm}(A) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi).$$

Reduction to the 01-Permanent

$$\mathrm{Perm}(A) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi).$$

A has entries -1, 0, and 1.

Reduction to the 01-Permanent

$$\mathrm{Perm}(A) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi).$$

A has entries -1, 0, and 1.
Goal: Reduce to instances with entries 0 and 1.

Reduction to the 01-Permanent

$$\mathrm{Perm}(A) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi).$$

A has entries -1, 0, and 1.
Goal: Reduce to instances with entries 0 and 1.

Classical Strategy: Use "interpolation" to recover #Sat(φ).

Reduction to the 01-Permanent

$$\mathrm{Perm}(A) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi).$$

A has entries -1, 0, and 1.
Goal: Reduce to instances with entries 0 and 1.

Classical Strategy: Use "interpolation" to recover #Sat(φ).
$\Rightarrow$ ETH is false if permanent is in time $2^{o(n/\log n)}$.

Reduction to the 01-Permanent

$$\mathrm{Perm}(A) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi).$$

A has entries -1, 0, and 1.
Goal: Reduce to instances with entries 0 and 1.

Classical Strategy: Use "interpolation" to recover $\#\mathrm{Sat}(\varphi)$.
$\Rightarrow$ ETH is false if permanent is in time $2^{o(n/\log n)}$.

Our Strategy: Use the isolation lemma and compute modulo 3.

Reduction to the 01-Permanent

$$\mathrm{Perm}(A) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi).$$

A has entries -1, 0, and 1.
Goal: Reduce to instances with entries 0 and 1.

Classical Strategy: Use "interpolation" to recover #Sat(φ).
$\Rightarrow$ ETH is false if permanent is in time $2^{o(n/\log n)}$.

Our Strategy: Use the isolation lemma and compute modulo 3.
$\Rightarrow$ ETH is false if permanent is in time $2^{o(n)}$.

Our strategy: Isolation

$$\mathrm{Perm}(G) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi)$$

Our strategy: Isolation

$$\mathrm{Perm}(G) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi)$$

If φ has at most one satisfying assignment, this gives:

$\mathrm{Perm}(G) \in \{0, (-2)^{c \cdot n}\}$.

Our strategy: Isolation

$$\mathrm{Perm}(G) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi)$$

If φ has at most one satisfying assignment, this gives:

$\mathrm{Perm}(G) \in \{0, (-2)^{c \cdot n}\}$.

Observation: $(-2)^{cn} \not\equiv 0 \mod 3$.

Our strategy: Isolation

$$\mathrm{Perm}(G) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi)$$

If φ has at most one satisfying assignment, this gives:

$$\mathrm{Perm}(G) \in \{0, (-2)^{c \cdot n}\}.$$

Observation: $(-2)^{cn} \not\equiv 0 \mod 3$.

We replace weights -1 with 2 and interpret the result modulo 3.

Our strategy: Isolation

$$\mathrm{Perm}(G) = (-2)^{c \cdot n} \cdot \#\mathrm{Sat}(\varphi)$$

If φ has at most one satisfying assignment, this gives:

$$\mathrm{Perm}(G) \in \{0, (-2)^{c \cdot n}\}.$$

Observation: $(-2)^{cn} \not\equiv 0 \mod 3$.

We replace weights -1 with 2 and interpret the result modulo 3.

"Isolated" formulas φ are still hard under ETH.

Isolation Lemma

Isolation Lemma

U3SAT

Given φ with ≤ 1 satisfying assignment.

Compute whether φ is satisfiable.

Isolation Lemma

U3SAT

Given φ with ≤ 1 satisfying assignment.

Compute whether φ is satisfiable.

U3SAT is NP-hard under randomized polytime reductions.
[Valiant, Vazirani '86]

Isolation Lemma

U3SAT

Given φ with ≤ 1 satisfying assignment.

Compute whether φ is satisfiable.

U3SAT is NP-hard under randomized polytime reductions.
[Valiant, Vazirani '86]

U3SAT is not computable in time $2^{o(n)}$ unless ETH is false.
[Calabro, Impagliazzo, Kabanets, Paturi '08]

Permanent Results Review

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)} \quad \text{with } d! \leq 2^{d \log d} \text{ terms.}$$

Permanent Results Review

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)} \quad \text{with } d! \leq 2^{d \log d} \text{ terms.}$$

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j} \quad \text{with } 2^d \text{ terms.}$$

Permanent Results Review

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)} \quad \text{with } d! \leq 2^{d \log d} \text{ terms.}$$

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j} \quad \text{with } 2^d \text{ terms.}$$

If Perm has $2^{o(d)}$-size formula exists, then ETH is false.

Permanent Results Review

$$\mathrm{Perm}(A) = \sum_{\pi \in S_d} \prod_{i=1}^{d} A_{i,\pi(i)} \quad \text{with } d! \leq 2^{d \log d} \text{ terms.}$$

$$\mathrm{Perm}(A) = \sum_{S \subseteq [d]} (-1)^{d-|S|} \prod_{i=1}^{d} \sum_{j \in S} A_{i,j} \quad \text{with } 2^d \text{ terms.}$$

If Perm has $2^{o(d)}$-size formula exists, then ETH is false.

We don't know any barriers for 1.99^d-size formulas!

Strong Exponential Time Hypothesis

A tool for exploring tighter bounds

Strong Exponential Time Hypothesis

A tool for exploring tighter bounds

$$c_k = \inf \left\{ \quad c \geq 1 \quad \middle| \quad k\text{-SAT} \text{ in time } c^n \quad \right\}$$

Strong Exponential Time Hypothesis

A tool for exploring tighter bounds

$$c_k = \inf \left\{ \quad c \geq 1 \quad \middle| \quad k\text{-SAT in time } c^n \quad \right\}$$

(SETH) $\lim_{k \to \infty} c_k = 2$

Strong Exponential Time Hypothesis

A tool for exploring tighter bounds

$$c_k = \inf \left\{ \quad c \geq 1 \quad \middle| \quad k\text{-SAT in time } c^n \quad \right\}$$

(SETH) $\lim_{k \to \infty} c_k = 2$

...it seems difficult to show that 1.99^d-algorithms for the permanent imply that SETH fails

Part II:
The Tutte Polynomial

The Tutte Polynomial

of a graph $G = (V, E)$

The Tutte Polynomial

of a graph $G = (V, E)$

$$T(G; x, y) := \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} \cdot (y-1)^{k(A)+|A|-|V|}$$

The Tutte Polynomial

of a graph $G = (V, E)$

$$T(G; x, y) := \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} \cdot (y-1)^{k(A)+|A|-|V|}$$

$k(A) =$ # connected components of the graph (V, A).

The Tutte Polynomial

of a graph $G = (V, E)$

$$T(G; x, y) := \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} \cdot (y-1)^{k(A)+|A|-|V|}$$

$k(A) =$ # connected components of the graph (V, A).

- $T(G; 1, 1) =$ # spanning trees in G.

The Tutte Polynomial

of a graph $G = (V, E)$

$$T(G; x, y) := \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} \cdot (y-1)^{k(A)+|A|-|V|}$$

$k(A) =$ # connected components of the graph (V, A).

- $T(G; 1, 1) =$ # spanning trees in G.
- $T(G; 2, 1) =$ # forests.

The Tutte Polynomial

of a graph $G = (V, E)$

$$T(G; x, y) := \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} \cdot (y-1)^{k(A)+|A|-|V|}$$

$k(A) =$ # connected components of the graph (V, A).

- $T(G; 1, 1) =$ # spanning trees in G.
- $T(G; 2, 1) =$ # forests.
- $T(G; 1-\lambda, 0) =$ # proper vertex-colourings with λ colours.

The Tutte Polynomial

of a graph $G = (V, E)$

$$T(G; x, y) := \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} \cdot (y-1)^{k(A)+|A|-|V|}$$

$k(A) =$ # connected components of the graph (V, A).

- $T(G; 1, 1) =$ # spanning trees in G.
- $T(G; 2, 1) =$ # forests.
- $T(G; 1-\lambda, 0) =$ # proper vertex-colourings with λ colours.
- $T(G; 1-\lambda, 0) = \chi(G; \lambda) =$ the chromatic polynomial.

The Tutte Polynomial

of a graph $G = (V, E)$

$$T(G; x, y) := \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} \cdot (y-1)^{k(A)+|A|-|V|}$$

$k(A) =$ # connected components of the graph (V, A).

- $T(G; 1, 1) =$ # spanning trees in G.
- $T(G; 2, 1) =$ # forests.
- $T(G; 1-\lambda, 0) =$ # proper vertex-colourings with λ colours.
- $T(G; 1-\lambda, 0) = \chi(G; \lambda) =$ the chromatic polynomial.
- $T(G; 0, 1-\lambda) =$ the nowhere-zero flow polynomial.

The Tutte Polynomial

of a graph $G = (V, E)$

$$T(G; x, y) := \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} \cdot (y-1)^{k(A)+|A|-|V|}$$

$k(A) =$ # connected components of the graph (V, A).

- $T(G; 1, 1) =$ # spanning trees in G.
- $T(G; 2, 1) =$ # forests.
- $T(G; 1-\lambda, 0) =$ # proper vertex-colourings with λ colours.
- $T(G; 1-\lambda, 0) = \chi(G; \lambda) =$ the chromatic polynomial.
- $T(G; 0, 1-\lambda) =$ the nowhere-zero flow polynomial.
- the Ising model, the Potts model, ...

Rough Complexity of the Tutte Polynomial

Definition is a formula of size 2^m.

Rough Complexity of the Tutte Polynomial

Definition is a formula of size 2^m.

Inclusion–Exclusion formula of size $2^{O(n)}$. [Björklund *et al.* '08]

Rough Complexity of the Tutte Polynomial

Definition is a formula of size 2^m.

Inclusion–Exclusion formula of size $2^{O(n)}$. [Björklund *et al.* '08]

No $\mathrm{poly}(n)$-size formula unless $\mathrm{P} = \mathrm{NP}$. [Jaeger, Vertigan, Welsh '90]

Rough Complexity of the Tutte Polynomial

Definition is a formula of size 2^m.

Inclusion–Exclusion formula of size $2^{O(n)}$. [Björklund *et al.* '08]

No $\mathrm{poly}(n)$-size formula unless $\mathrm{P} = \mathrm{NP}$. [Jaeger, Vertigan, Welsh '90]

No $2^{o(n)}$-size formula unless ETH is false. [we]

Rough Complexity of the Tutte Polynomial

Definition is a formula of size 2^m.

Inclusion–Exclusion formula of size $2^{O(n)}$. [Björklund *et al.* '08]

No $\mathrm{poly}(n)$-size formula unless $\mathrm{P} = \mathrm{NP}$. [Jaeger, Vertigan, Welsh '90]

No $2^{o(n)}$-size formula unless ETH is false. [we]

Open: 1.99^n

Rough Complexity of the Tutte Polynomial

Definition is a formula of size 2^m.

Inclusion–Exclusion formula of size $2^{O(n)}$. [Björklund *et al.* '08]

No $\text{poly}(n)$-size formula unless $\text{P} = \text{NP}$. [Jaeger, Vertigan, Welsh '90]

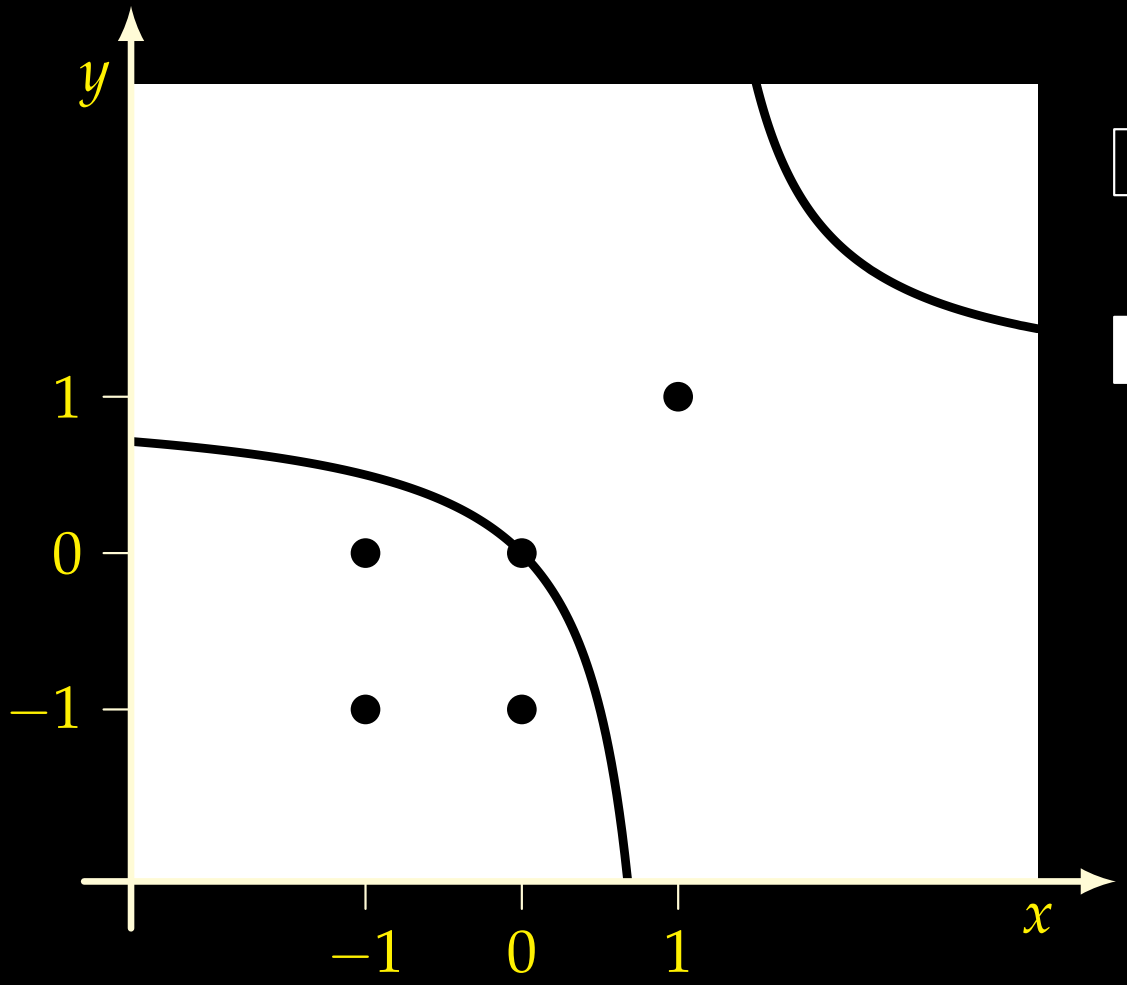
No $2^{o(n)}$-size formula unless ETH is false. [we]

Open: 1.99^n

For fixed (x, y), compute the number $T(G; x, y)$ given G

The Tutte Plane under P $\neq$ NP

[Jaeger, Vertigan, Welsh '90]

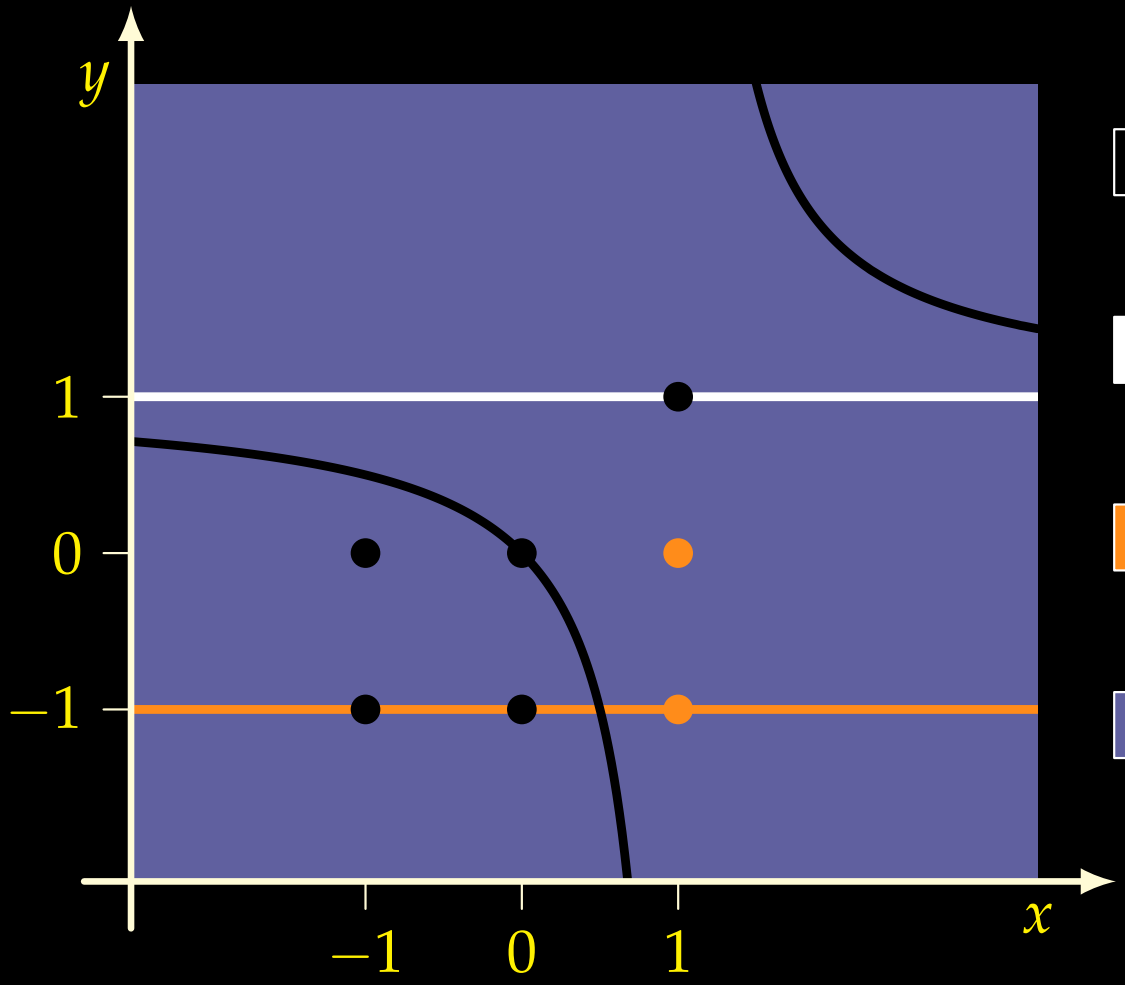


poly(n)

Not poly(n)

The Tutte Plane under ETH

for Multigraphs [we]



poly(n)

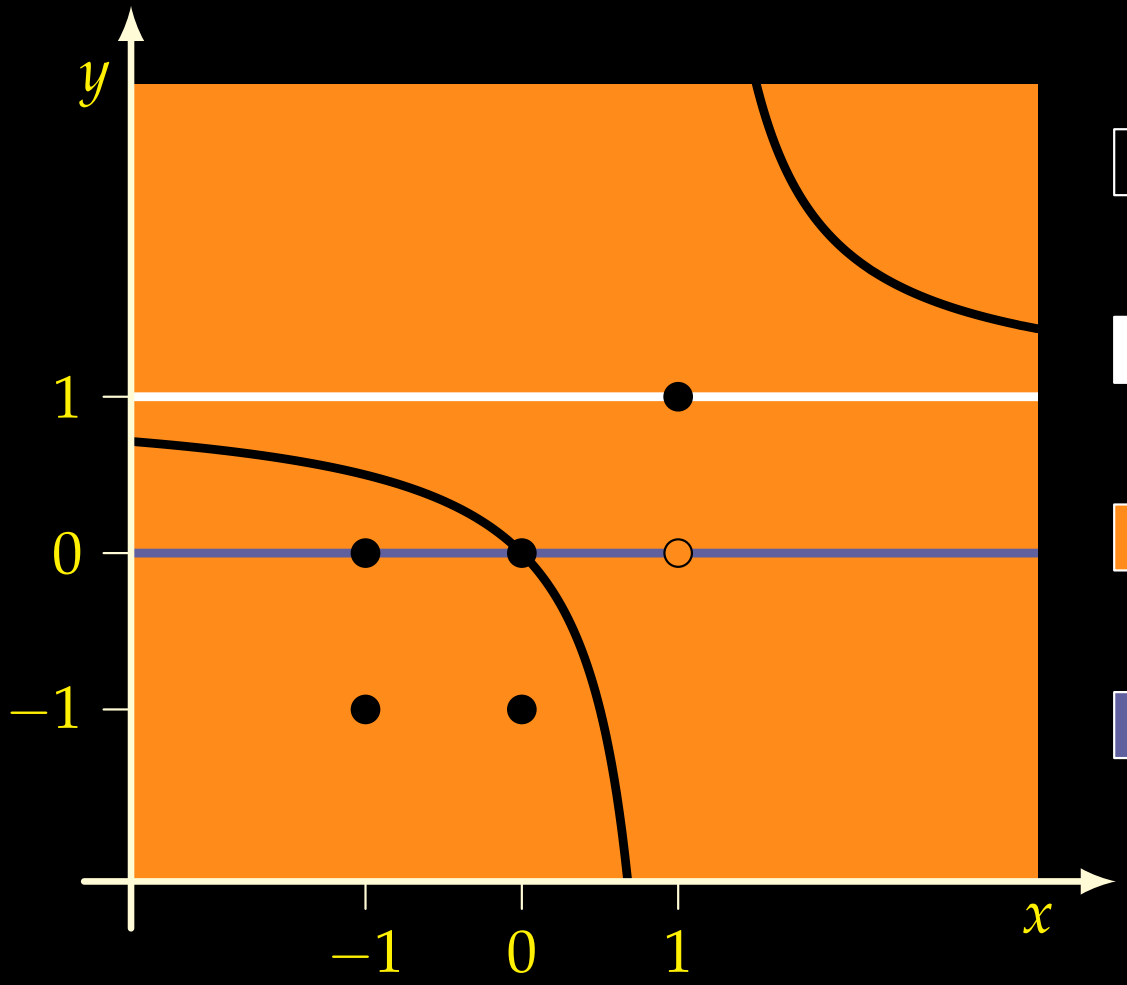
Not poly(n)

Not $2^{o(n/\log^3 n)}$

Not $2^{o(n)}$

The Tutte Plane under ETH

for Simple Graphs [we]



poly(n)

Not poly(n)

Not $2^{o(n/\log^3 n)}$

Not $2^{o(n)}$

Tutte Plane Hardness Proofs

Tutte Plane Hardness Proofs

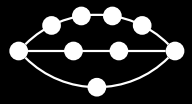
Uses Brylawski's theorem (1980) for the Tutte polynomial.

Tutte Plane Hardness Proofs

Uses Brylawski's theorem (1980) for the Tutte polynomial.
That is, replace every edge (u, v) with...

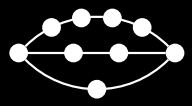
Tutte Plane Hardness Proofs

Uses Brylawski's theorem (1980) for the Tutte polynomial.
That is, replace every edge (u, v) with...

(1) Theta graphs: u  v

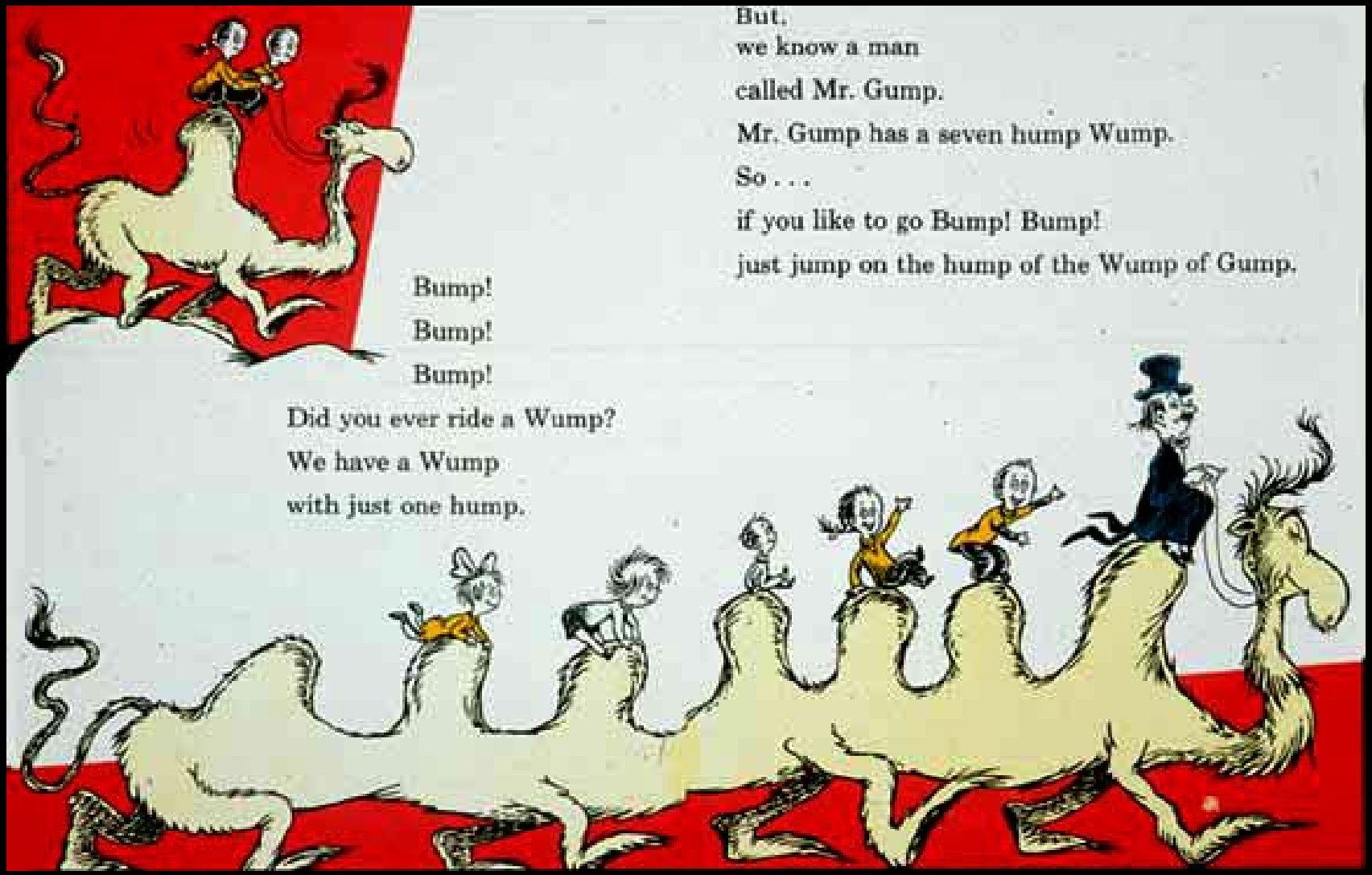
Tutte Plane Hardness Proofs

Uses Brylawski's theorem (1980) for the Tutte polynomial.
That is, replace every edge (u, v) with...

(1) Theta graphs: u  v

or (2) Wump graphs:

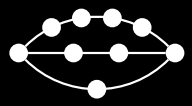
Wumps



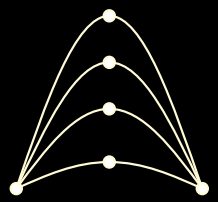
Dr. Seuss (1960): One Fish Two Fish Red Fish Blue Fish

Tutte Plane Hardness Proofs

Uses Brylawski's theorem (1980) for the Tutte polynomial.
That is, replace every edge (u, v) with...

(1) Theta graphs: u  v

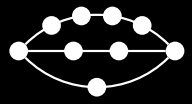
or (2) Wump graphs:



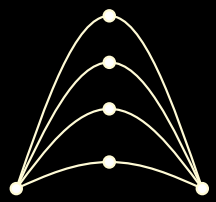
$(4, 2)$-hump

Tutte Plane Hardness Proofs

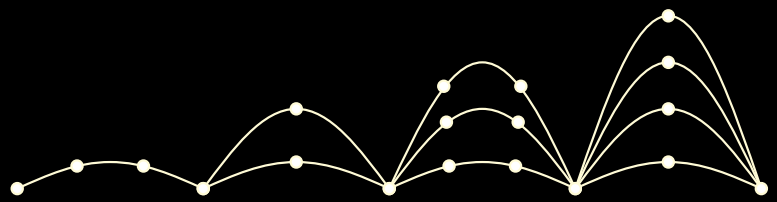
Uses Brylawski's theorem (1980) for the Tutte polynomial.
That is, replace every edge (u, v) with...

(1) Theta graphs: u  v

or (2) Wump graphs:



$(4, 2)$-hump

u  v

$\langle 3, 2, 3, 2 \rangle$-Wump

Results Overview

Results Overview

The exponential time hypothesis (ETH) is **false** if:

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- $\mathrm{Perm}\big((d \times d)\text{-matrix } A\big)$ can be computed in time $2^{o(d)}$.

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- $\mathrm{Perm}\big((d \times d)\text{-matrix } A\big)$ can be computed in time $2^{o(d)}$.
- The Tutte polynomial can be computed in time $2^{o(n)}$.

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- Perm$\big((d \times d)$-matrix $A\big)$ can be computed in time $2^{o(d)}$.
- The Tutte polynomial can be computed in time $2^{o(n)}$.
- #2-SAT can be computed in time $2^{o(n)}$.

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- Perm$\big((d \times d)$-matrix $A\big)$ can be computed in time $2^{o(d)}$.
- The Tutte polynomial can be computed in time $2^{o(n)}$.
- #2-SAT can be computed in time $2^{o(n)}$.

Open:

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- Perm$\big((d \times d)$-matrix $A\big)$ can be computed in time $2^{o(d)}$.
- The Tutte polynomial can be computed in time $2^{o(n)}$.
- #2-SAT can be computed in time $2^{o(n)}$.

Open:

- 1.99^d-time algorithms (or formulas) for the permanent or the Tutte polynomial.

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- Perm$\big((d \times d)$-matrix $A\big)$ can be computed in time $2^{o(d)}$.
- The Tutte polynomial can be computed in time $2^{o(n)}$.
- #2-SAT can be computed in time $2^{o(n)}$.

Open:

- 1.99^d-time algorithms (or formulas) for the permanent or the Tutte polynomial. (even modulo 3)

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- Perm$\big((d \times d)$-matrix $A\big)$ can be computed in time $2^{o(d)}$.
- The Tutte polynomial can be computed in time $2^{o(n)}$.
- #2-SAT can be computed in time $2^{o(n)}$.

Open:

- 1.99^d-time algorithms (or formulas) for the permanent or the Tutte polynomial. (even modulo 3)
- Get tight results for the whole Tutte plane.

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- Perm$\big((d \times d)$-matrix $A\big)$ can be computed in time $2^{o(d)}$.
- The Tutte polynomial can be computed in time $2^{o(n)}$.
- #2-SAT can be computed in time $2^{o(n)}$.

Open:

- 1.99^d-time algorithms (or formulas) for the permanent or the Tutte polynomial. (even modulo 3)
- Get tight results for the whole Tutte plane.
- Prove hardness under the strong ETH.

Results Overview

The exponential time hypothesis (ETH) is **false** if:

- Perm$\big((d \times d)$-matrix $A\big)$ can be computed in time $2^{o(d)}$.
- The Tutte polynomial can be computed in time $2^{o(n)}$.
- #2-SAT can be computed in time $2^{o(n)}$.

Open:

- 1.99^d-time algorithms (or formulas) for the permanent or the Tutte polynomial. (even modulo 3)
- Get tight results for the whole Tutte plane.
- Prove hardness under the strong ETH.
- Does a Toda Theorem ($\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$) hold for #ETH or #M[1]?

Possible Toda Theorems

Possible Toda Theorems

k-SAT Given φ, decide $\exists x.\varphi(x) = 1$

Possible Toda Theorems

k-SAT Given φ, decide $\exists x.\varphi(x) = 1$

Π_2 k-CNF Given φ, decide $\forall x_1.\exists x_2.\varphi(x_1, x_2) = 1$

Possible Toda Theorems

k-SAT Given φ, decide $\exists x.\varphi(x) = 1$

Π_2 k-CNF Given φ, decide $\forall x_1.\exists x_2.\varphi(x_1, x_2) = 1$
(related to the polynomial time hierarchy)

Possible Toda Theorems

k-SAT Given φ, decide $\exists x.\varphi(x) = 1$

Π_2 k-CNF Given φ, decide $\forall x_1.\exists x_2.\varphi(x_1, x_2) = 1$
(related to the polynomial time hierarchy)

1 Does ¬#ETH imply that Π_2 k-CNF is in time $2^{o(n)}$?

Possible Toda Theorems

k-SAT Given φ, decide $\exists x.\varphi(x) = 1$

Π_2 k-CNF Given φ, decide $\forall x_1.\exists x_2.\varphi(x_1, x_2) = 1$
(related to the polynomial time hierarchy)

1 Does ¬#ETH imply that Π_2 k-CNF is in time $2^{o(n)}$?

2 Does ¬#ETH imply that CNF-SAT is in time $2^{o(n)}$?